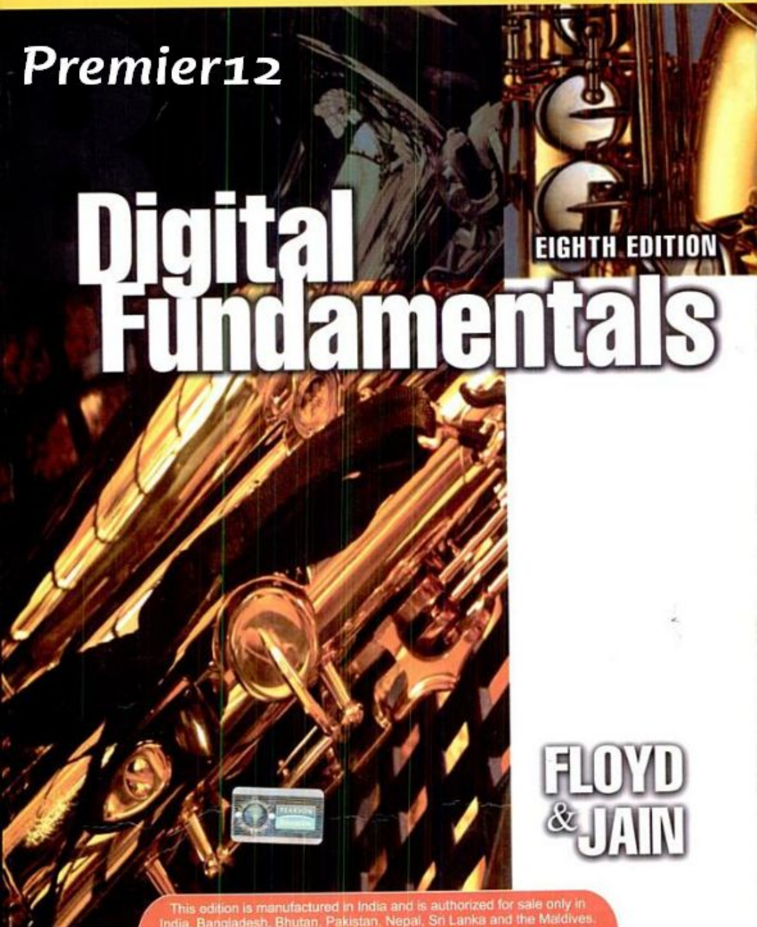
PEARSON
Education

Premier12



Digital Fundamentals

EIGHTH EDITION

FLOYD
& JAIN

This edition is manufactured in India and is authorized for sale only in India, Bangladesh, Bhutan, Pakistan, Nepal, Sri Lanka and the Maldives. Circulation of this edition outside of these territories is UNAUTHORIZED.

VISIT…

LANZAROTE
Caliente.COM

Authorized adaptation from the United States edition, entitled *Digital Fundamentals, 8th Edition,*
ISBN: 9780130942005 by Floyd, Thomas L., published by Pearson Education. Inc., Copyright © 2003

Indian Subcontinent Adaptation
Copyright © 2006 Dorling Kindersley (India) Pvt. Ltd

Copyright © 2005 by Pearson Education (Singapore) Pte. Ltd
This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, resold, hired out or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than that in which it is published without a similar condition including this condition being imposed on subsequent purchaser and without limiting the rights under copyright reserved above, no part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), without the prior written permission of both the copyright owner and above-mentioned publisher of this book.

ISBN 978-81-7758-763-0

First Impression, 2006
Second Impression, 2007
Third Impression
Fourth Impression, 2008
Fifth Impression
Sixth Impression
Seventh Impression, 2009
Eighth Impression, 2009

This edition is manufactured in India and is authorized for sale only in India, Bangladesh, Bhutan, Pakistan, Nepal, Sri Lanka and the Maldives.

Published by Dorling Kindersley (India) Pvt. Ltd., licensees of Pearson Education in South Asia.

Head Office: 482 FIE, Patparganj, Delhi 110 092, India.
Registered Office: 14 Local Shopping Centre, Panchsheel Park, New Delhi 110 017, India.

Printed in India at Anand Sons.

Brief Contents

1 Introductory Digital Concepts
2 Number Systems, Operations, and Codes 16
3 Logic Gates 78
4 Boolean Algebra and Logic Simplification 118
5 Combinational Logic 172
6 Functions of Combinational Logic 202
7 Flip-Flops 266
8 Counters 304
9 Shift Registers 354
10 Memory and Storage 390
11 Integrated Circuit Technologies 452
12 Programmable Logic Devices (PLDs) 502
Answers to Odd-Numbered Problems 552
Glossary 570
Index 579

Contents

1 Introductory Digital Concepts

Chapter Objectives
Introduction
1–1 Digital and Analog Quantities
1–2 Binary Digits, Logic Levels, and Digital Waveforms 3
1–3 Introduction to Basic Logic Operations 9
1–4 Digital Integrated Circuits 11
Summary 12
Self-Test 12
Problems 13
Answers 15
Section Reviews 15
Supplementary Problems for Examples 15
Self-Test 15

2 Number Systems, Operations, and Codes 16

Chapter Objectives
Introduction
2–1 Decimal Numbers 16
2–2 Binary Numbers 18
2–3 Decimal-to-Binary Conversion 21
2–4 Binary Arithmetic 23
2–5 1's and 2's Complements of Binary Numbers 27
2–6 Signed Numbers 29
2–7 Arithmetic Operations with Signed Numbers 35
2–8 Hexadecimal Numbers 42
2–9 Octal Numbers 48
2–10 Binary Coded Decimal (BCD) 51
2–11 Digital Codes 54
2–12 Error Detection and Correction Codes 61
Summary 69
Self-Test 70
Problems 71
Answers 75
Section Reviews 75
Supplementary Problems for Examples 77
Self-Test 77

3 Logic Gates 78

Chapter Objectives
Introduction
3–1 The Inverter 78
3–2 The AND Gate 82
3–3 The OR Gate 88
3–4 The NAND Gate 93
3–5 The NOR Gate 98
3–6 The Exclusive-OR and Exclusive-NOR Gates 103
3–7 Examples of IC Gates 107
Summary 109
Self-Test 110
Problems 112
Answers 114
Section Reviews 114
Supplementary Problems for Examples 115
Self-Test 117

4 Boolean Algebra and Logic Simplification 118

Chapter Objectives
Introduction
4–1 Boolean Operations and Expressions 118
4–2 Laws and Rules of Boolean Algebra 120
4–3 DeMorgan's Theorems 125
4–4 Boolean Analysis of Logic Circuits 129
4–5 Simplification Using Boolean Algebra 131
4–6 Standard Forms of Boolean Expressions 134
4–7 Boolean Expressions and Truth Tables 140
4–8 The Karnaugh Map 144
4–9 Karnaugh Map SOP Minimization 146
4–10 Karnaugh Map POS Minimization 155

4–11 Five-Variable Karnaugh Maps 159
Summary 162
Self-Test 162
Problems 163
Answers 169
Section Reviews 169
Supplementary Problems for Examples 170
Self-Test 171

5 Combinational Logic 172

Chapter Objectives
Introduction
5–1 Basic Combinational Logic Circuits 172
5–2 Implementing Combinational Logic 177
5–3 The Universal Property of NAND and NOR Gates 183
5–4 Combinational Logic Using NAND and NOR Gates 185
5–5 Logic Circuit Operation with Pulse Waveforms 191
Summary 194
Self-Test 195
Problems 196
Answers 200
Section Reviews 200
Supplementary Problems for Examples 201
Self-Test 201

6 Functions of Combinational Logic 202

Chapter Objectives
Introduction
6–1 Basic Overview of Logic Functions 202
6–2 Basic Adders 208
6–3 Parallel Binary Adders 211
6–4 Comparators 218
6–5 Decoders 222
6–6 Encoders 231
6–7 Code Converters 236
6–8 Multiplexers (Data Selectors) 238
6–9 Demultiplexers 247
6–10 Parity Generators/Checkers 248
6–11 Glitches in Decoder Circuits 251
Summary 253
Self-Test 255
Problems 256
Answers 261
Section Reviews 261
Supplementary Problems for Examples 263
Self-Test 265

7 Flip-Flops 266

Chapter Objectives
Introduction
7–1 Latches 266
7–2 Edge-Triggered Flip-Flops 273
7–3 Master-Slave Flip-Flops 285
7–4 Flip-Flop Operating Characteristics 287
7–5 Flip-Flop Applications 290
Summary 295
Self-Test 295
Problems 296
Answers 301
Section Reviews 301
Supplementary Problems for Examples 302
Self-Test 303

8 Counters 304

Chapter Objectives
Introduction
8–1 Asynchronous Counter Operation 305
8–2 Synchronous Counter Operation 313
8–3 Up/Down Synchronous Counters 320
8–4 Design of Synchronous Counters 324
8–5 Cascaded Counters 333
8–6 Counter Decoding 336
8–7 Counter Applications 340
Summary 345
Self-Test 346
Problems 347
Answers 351
Section Reviews 351
Supplementary Problems for Examples 352
Self-Test 353

9 Shift Registers 354

Chapter Objectives
Introduction
9–1 Basic Shift Register Functions 354
9–2 Serial In/Serial Out Shift Registers 356
9–3 Serial In/Parallel Out Shift Registers 360
9–4 Parallel In/Serial Out Shift Registers 363
9–5 Parallel In/Parallel Out Shift Registers 366
9–6 Bidirectional Shift Registers 368
9–7 Shift Register Counters 371
9–8 Shift Register Applications 375
Summary 382
Self-Test 382
Problems 383
Answers 387
- *Section Reviews 387*
- *Supplementary Problems for Examples 388*
- *Self-Test 389*

10 Memory and Storage 390

Chapter Objectives
Introduction
10–1 Basics of Semiconductor Memory 390
10–2 Random-Access Memories (RAMs) 394
10–3 Read-Only Memories (ROMs) 407
10–4 Programmable ROMs (PROMs and EPROMs) 412
10–5 Flash Memories 416
10–6 Memory Expansion 420
10–7 Special Types of Memories 426
10–8 Magnetic and Optical Storage 432
10–9 Testing Memory Chips 438
Summary 443
Self-Test 444
Problems 445
Answers 450
- *Section Reviews 450*
- *Supplementary Problems for Examples 451*
- *Self-Test 451*

11 Integrated Circuit Technologies 452

Chapter Objectives
Introduction
11–1 Basics of Digital Integrated Circuits 453
11–2 Basic Operational Characteristics and Parameters 461
11–3 CMOS Circuits 470
11–4 TTL Circuits 475
11–5 Practical Considerations in the Use of TTL 480
11–6 Comparison of CMOS and TTL Performance 487
11–7 Emitter-Coupled Logic (ECL) Circuits 488
11–8 PMOS, NMOS, and E^2CMOS 489
Summary 492
Self-Test 492
Problems 493
Answers 499
- *Section Reviews 499*
- *Supplementary Problems for Examples 500*
- *Self-Test 501*

12 Programmable Logic Devices (PLDs) 502

Chapter Objectives
Introduction
12–1 Introduction to Programmable Logic Devices (PLDs) 502
12–2 Simple Programmable Logic Devices (SPLDs) 504
12–3 Programmable Array Logic (PAL) 507
12–4 Basic Concepts of GAL 513
12–5 Programming of SPLDs 517
12–6 The GAL22V10 520
12–7 The GAL16V8 529
12–8 Introduction to CPLDs 533
12–9 Introduction to FPGAs 537
Summary 543
Self-Test 544
Problems 546
Answers 549
- *Section Reviews 549*
- *Supplementary Problems for Examples 551*
- *Self-Test 551*

Answers to Odd-Numbered Problems 552

Glossary 570

Index 579

1

INTRODUCTORY DIGITAL CONCEPTS

CHAPTER OBJECTIVES

- Explain the basic differences between digital and analog quantities
- Show how voltage levels are used to represent digital quantities
- Describe various parameters of a pulse waveform such as rise time, fall time, pulse width, frequency, period, and duty cycle
- Explain the basic logic operations of NOT, AND, and OR
- Describe integrated circuit (IC), DIP, IC package for fixed function ICs, and pin numbering

INTRODUCTION

The term *digital* is derived from the way computers perform operations, by counting digits. For many years, applications of digital electronics were confined to computer systems. Today, digital technology is applied in a wide range of areas in addition to computers. Such applications as television, communications systems, radar, navigation and guidance systems, military systems, medical instrumentation, industrial process control, and consumer electronics use digital techniques. Digital technology has progressed from vacuum-tube circuits to discrete transistors to complex integrated circuits, some of which contain millions of transistors.

This chapter introduces you to digital electronics and provides a broad overview of many important concepts.

1–1 DIGITAL AND ANALOG QUANTITIES

Electronic circuits can be divided into two broad categories: digital and analog. Digital electronics involves quantities with discrete values, and analog electronics involves quantities with continuous values. Although you will be studying digital fundamentals in this book, you should also know about analog because many applications require both.

After completing this section, you should be able to

- Define *analog*
- Define *digital*
- Explain the difference between digital and analog quantities
- State the advantages of digital over analog
- Give examples of how digital and analog quantities are used in electronics

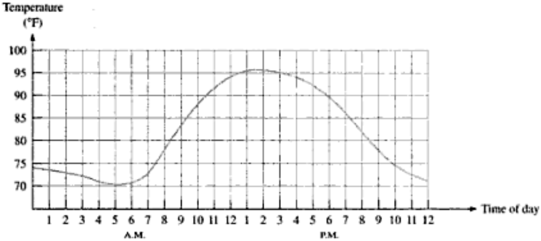
An **analog*** quantity is one having continuous values. A **digital** quantity is one having a discrete set of values. Most things that can be measured quantitatively appear in nature in analog form. For example, the air temperature changes over a continuous range of values. During a given day, the temperature does not go from, say, 70° to 71° instantaneously; it takes on all the infinite values in between. If you graphed the temperature on a typical summer day, you would have a smooth, continuous curve similar to the curve in Figure 1–1. Other examples of analog quantities are time, pressure, distance, and sound.

---

*All bold terms are important and are defined in the end-of-book glossary.

FIGURE 1–1

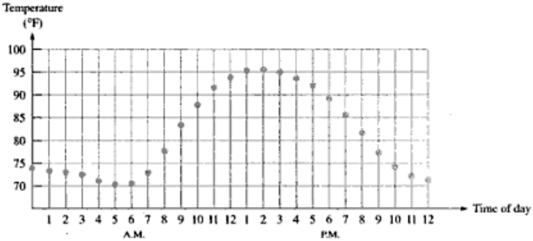
Graph of an analog quantity (temperature versus time)



Rather than graphing the temperature on a continuous basis, suppose you just take a temperature reading every hour. Now, you have sampled values representing the temperature at discrete points in time (every hour) over a 24-hour period, as indicated in Figure 1–2. You have effectively converted an analog quantity to a form that can now be digitized by representing each sampled value by a digital code. It is important to realize that Figure 1–2 itself is not the digital representation of the analog quantity. In the digital representation each dot will be represented by a series of 1s and 0s.

FIGURE 1–2

Sampled-value representation (quantization) of the analog quantity in Figure 1–1



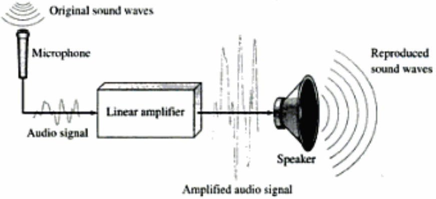
The Digital Advantage Digital representation has certain advantages over analog representation in electronics applications. For one thing, digital data can be processed and transmitted more efficiently and reliably than analog data. Also, digital data has a great advantage when storage is necessary. For example, music when converted to digital form can be stored more compactly and reproduced with greater accuracy and clarity than is possible when it is in analog form. Noise (unwanted voltage fluctuations) does not affect digital data nearly as much as it does analog signals.

An Analog Electronic System

A public address system, used to amplify sound so that it can be heard by a large audience, is one example of an application of analog electronics. The basic diagram in Figure 1–3 illustrates that sound waves, which are analog in nature, are picked up by a microphone and converted to a small analog voltage called the audio signal. This voltage varies continuously as the volume and frequency of the sound changes and is applied to the input of a linear amplifier. The output of the amplifier, which is an increased reproduction of input voltage, goes to the speaker(s). The speaker changes the amplified audio signal back to sound waves that have a much greater volume than the original sound waves picked up by the microphone.

▸ **FIGURE 1–3**

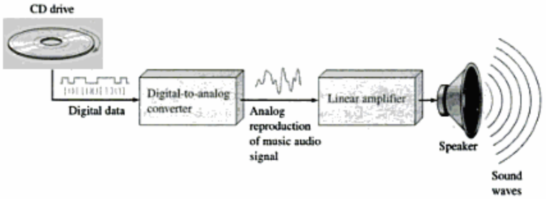
A basic audio public address system


Original sound waves
Microphone
Audio signal
Linear amplifier
Reproduced sound waves
Speaker
Amplified audio signal

A System Using Digital and Analog Methods

The compact disk (CD) player is an example of a system in which both digital and analog circuits are used. The simplified diagram in Figure 1–4 illustrates the basic principle of a CD player.

Music in digital form is stored on the compact disk. A laser diode optical system picks up the digital data from the rotating disk and transfers it to the **digital-to-analog converter (DAC)**. The DAC changes the digital data into an analog signal, that is an electrical reproduction of the original music. This signal is amplified and sent to the speaker for you to enjoy. When the music was originally recorded on the CD, a process, essentially the reverse of the one described here, using an **analog-to-digital converter (ADC)** was used.


CD drive
Digital data
Digital-to-analog converter
Analog reproduction of music audio signal
Linear amplifier
Speaker
Sound waves

▲ **FIGURE 1–4**

Basic principle of a CD player

SECTION 1-1 REVIEW

Answers are at the end of the chapter.

1. Define *analog.*
2. Define *digital.*
3. Explain the difference between a digital quantity and an analog quantity.
4. Give an example of a system that is analog and one that is a combination of both digital and analog. Name a system that is entirely digital.

1–2 BINARY DIGITS, LOGIC LEVELS, AND DIGITAL WAVEFORMS

Digital electronics involves circuits and systems in which there are only two possible states. These states are represented by two different voltage levels: A HIGH and a LOW. The two states can also be represented by current levels, open and closed switches, or lamps turned on and off. In digital systems such as computers, combinations of the two states, called *codes*, are used to represent numbers, symbols, alphabetic characters, and other types of information. The two-state number system is called *binary*, and its two digits are 0 and 1. A binary digit is called a *bit*.

After completing this section, you should be able to

■ Define *binary* ■ Define *bit* ■ Name the bits in a binary system ■ Explain how voltage levels are used to represent bits ■ Explain how voltage levels are interpreted by a digital circuit ■ Describe the general characteristics of a pulse ■ Determine the amplitude, rise time, fall time, and width of a pulse ■ Identify and describe the characteristics of a digital waveform ■ Determine the amplitude, period, frequency, and duty cycle of a digital waveform ■ Explain what a timing diagram is and state its purpose ■ Explain serial and parallel data transfer and state the advantage and disadvantage of each

Binary Digits

The two digits in the binary system, 1 and 0, are called **bits**, which is a contraction of the words *binary digit*. In digital circuits, two different voltage levels are used to represent the two bits. Generally, 1 is represented by the higher voltage, which we will refer to as a HIGH, and a 0 is represented by the lower voltage level, which we will refer to as a LOW. This is called **positive logic** and will be used throughout the book.

HIGH = 1 and **LOW = 0**

Another system in which a 1 is represented by a LOW and a 0 is represented by a HIGH is called *negative logic*.

Groups of bits (combinations of 1s and 0s), called *codes*, are used to represent numbers, letters, symbols, instructions, and anything else required in a given application.

Logic Levels

The voltages used to represent a 1 and a 0 are called *logic* levels. Ideally, one voltage level represents a HIGH and another voltage level represents a LOW. In a practical digital circuit, however, a HIGH can be any voltage between a specified minimum value and a specified maximum value. Likewise, a LOW can be any voltage between a specified minimum and a specified maximum. There can be no overlap between the accepted HIGH levels and the accepted LOW levels.

FIGURE 1–5

Logic level ranges of voltage for a digital circuit

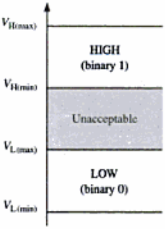


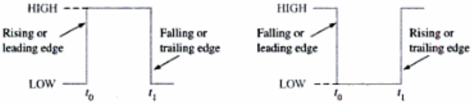
Figure 1–5 illustrates the general range of LOWs and HIGHs for a digital circuit. The variable $V_{H(max)}$ represents the maximum HIGH voltage value, and $V_{H(min)}$ represents the minimum HIGH voltage value. The maximum LOW voltage value is represented by $V_{L(max)}$, and the minimum LOW voltage value is represented by $V_{L(min)}$. The voltage values between $V_{L(max)}$ and $V_{H(min)}$ are unacceptable for proper operation. A voltage in the unacceptable range can appear as either a HIGH or a LOW to a given circuit. Therefore, these unacceptable values are never used. For example, the HIGH values for a certain type of digital circuit called TTL may range from 2 V to 5 V and the LOW values may range from 0 V to 0.8 V. So, for example, if a voltage of 3.5 V is applied, the circuit will accept it as a HIGH or binary 1. If a voltage of 0.5 V is applied, the circuit will accept it as a LOW or binary 0. For this type of circuit, voltages between 0.8 V and 2 V are unacceptable and are never used.

Digital Waveforms

Digital waveforms consist of voltage levels that are changing back and forth between the HIGH and LOW levels or states. Figure 1–6(a) shows that a single positive-going **pulse** is generated when the voltage (or current) goes from its normally LOW level to its HIGH level and then back to its LOW level. The negative-going pulse in Figure 1–6(b) is generated when the voltage goes from its normally HIGH level to its LOW level and back to its HIGH level. A digital waveform is made up of a series of pulses.

FIGURE 1–6

Ideal pulses



(a) Positive-going pulse

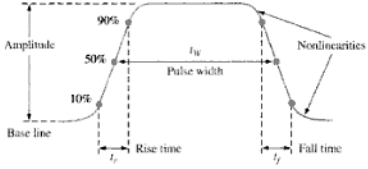
(b) Negative-going pulse

The Pulse As indicated in Figure 1–6, the pulse has two edges: a **leading edge** that occurs first at time t_0 and a **trailing edge** that occurs last at time t_1. For a positive-going pulse, the leading edge is a rising edge, and the trailing edge is a falling edge. The pulses in Figure 1–6 are ideal because the rising and falling edges are assumed to change in zero time (instantaneously). In practice, these transitions never occur instantaneously, although for most digital work you can assume ideal pulses.

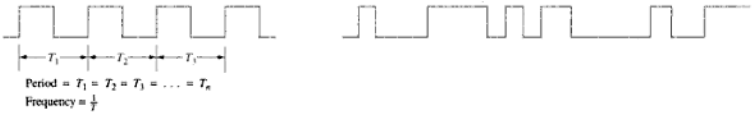
Figure 1–7 shows a nonideal pulse. The time required for the pulse to go from its LOW level to its HIGH level is called the **rise time** (t_r) and the time required for the transition from the HIGH level to the LOW level is called the **fall time** (t_f). In practice, it is common to measure rise time from 10% of the pulse **amplitude** (height from baseline) to 90% of the pulse amplitude and to measure the fall time from 90% to 10% of the pulse amplitude, as indicated in Figure 1–7. The bottom 10% and the top 10% of the pulse are not included in the rise and fall times because of the nonlinearities in the waveform in these areas. The **pulse width** (t_W) is a measure of the duration of the pulse and is often defined as the time interval between the 50% points on the rising and falling edges, as indicated in Figure 1–7.

▶FIGURE 1–7

Nonideal pulse characteristics



Waveform Characteristics Most waveforms encountered in digital systems are composed of series of pulses, sometimes called *pulse trains*, and can be classified as either periodic or nonperiodic. A **periodic** pulse waveform is one that repeats itself at a fixed interval, called a **period** (T). The **frequency** (f) is the rate at which it repeats itself and is measured in hertz (Hz). A nonperiodic pulse waveform, of course, does not repeat itself at fixed intervals and may be composed of pulses of randomly differing pulse widths and/or randomly differing time intervals between the pulses. An example of each type is shown in Figure 1–8.


T_1
T_2
T_3
Period = $T_1 = T_2 = T_3 = \ldots = T_n$
Frequency = $\frac{1}{T}$

(a) Periodic (square wave)

(b) Nonperiodic

▲FIGURE 1–8

Examples of digital waveforms

The frequency (f) of a pulse (digital) waveform is the reciprocal of the period. The relationship between frequency and period is expressed as follows:

$$f = \frac{1}{T}$$

Equation 1–1

$$T = \frac{1}{f}$$

Equation 1–2

An important characteristic of a periodic digital waveform is its duty cycle. The **duty cycle** is the ratio of the pulse width (t_W) to the period (T) and can be expressed as a percentage.

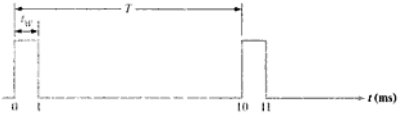
$$\text{Duty cycle} = \left(\frac{t_W}{T}\right)100\%$$

Equation 1–3

EXAMPLE 1–1

A portion of a periodic digital waveform is shown in Figure 1–9. The measurements are in milliseconds. Determine the following:

(a) period **(b)** frequency **(c)** duty cycle



▲ **FIGURE 1–9**

Solution **(a)** The period is measured from the edge of one pulse to the corresponding edge of the next pulse. In this case T is measured from leading edge to leading edge, as indicated. T equals **10 ms**.

(b) $f = \frac{1}{T} = \frac{1}{10\text{ ms}} = \mathbf{100\ Hz}$

(c) Duty cycle $= \left(\frac{t_W}{T}\right)100\% = \left(\frac{1\text{ ms}}{10\text{ ms}}\right)100\% = \mathbf{10\%}$

Supplementay Problem A periodic digital waveform has a pulse width of 25 μs and a period of 150 μs. Determine the frequency and the duty cycle.

A Digital Waveform Carries Binary Information

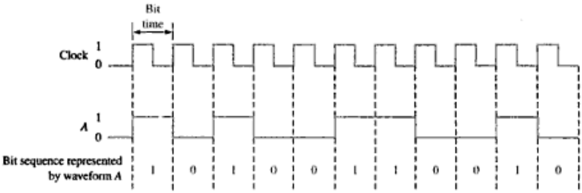
Binary information that is handled by digital systems appears as waveforms that represent sequences of bits. When the waveform is HIGH, a binary 1 is present; when the waveform is LOW, a binary 0 is present. Each bit in a sequence occupies a defined time interval called a **bit time**.

The Clock In digital systems, all waveforms are synchronized with a basic timing waveform called the **clock.** The clock is a periodic waveform in which each interval between pulses (the period) equals the time for one bit.

An example of a clock waveform is shown in Figure 1–10. Notice that, in this case, each change in level of waveform A occurs at the leading edge of the clock waveform. In other cases, level changes occur at the trailing edge of the clock. During each bit time of the

▲ **FIGURE 1–10**

Clock waveform synchronized with a waveform representation of a sequence of bits



clock, waveform *A* is either HIGH or LOW. These HIGHs and LOWs represent a sequence of bits as indicated. A group of several bits can be used as a piece of binary information, such as a number or a letter. The clock waveform itself does not carry information.

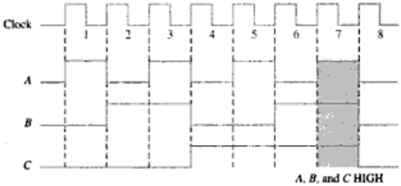
Timing Diagrams

A **timing diagram** is a graph of digital waveforms showing the actual time relationship of two or more waveforms and how each waveform changes in relation to the others. Figure 1–10 is an example of a simple timing diagram that shows how the clock waveform and waveform *A* are related on a time base.

By looking at a timing diagram, you can determine the states (HIGH or LOW) of all the waveforms at any specified point in time and the exact time that a waveform changes state relative to the other waveforms. Figure 1–11 is an example of a timing diagram made up of four waveforms. From this timing diagram you can see, for example, that the three waveforms *A*, *B*, and *C* are HIGH only during bit time 7 and they all change back LOW at the end of bit time 7 (shaded area).

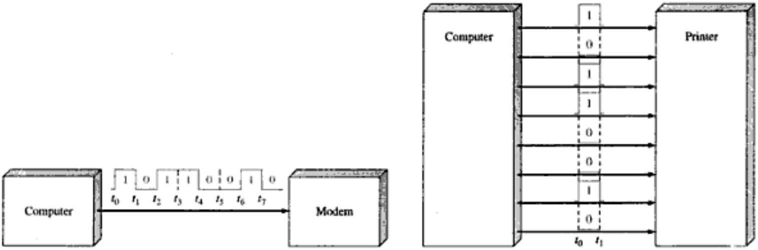
FIGURE 1–11

Example of a timing diagram



Data Transfer

Data refers to groups of bits that convey some type of information. Binary data, which are represented by digital waveforms, must be transferred from one circuit to another within a digital system or from one system to another in order to accomplish a given purpose. For example, numbers stored in binary form in the memory of a computer must be transferred to the computer's central processing unit in order to be added. The sum of the addition must then be transferred to a monitor for display and/or transferred back to the memory. In computer systems, as illustrated in Figure 1–12, binary data are transferred in two ways: serial and parallel. In the figure, only the data lines are shown.



(a) Serial transfer of 8 bits of binary data from computer to modem. Interval t_0 to t_1 is first

(b) Parallel transfer of 8 bits of binary data from computer to printer. The beginning time is t_0

FIGURE 1–12

When bits are transferred in **serial** form from one point to another, they are sent one bit at a time along a single conductor, as illustrated in Figure 1–12(a) for the case of a computer-to-modem transfer. During the time interval from t_0 to t_1, the first bit is transferred. During the time interval from t_1 to t_2, the second bit is transferred, and so on. To transfer eight bits in series, it takes eight time intervals.

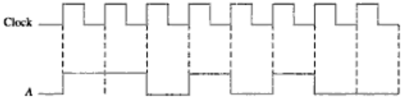
When bits are transferred in **parallel** form, all the bits in a group are sent out on separate lines at the same time. There is one line for each bit, as shown in Figure 1–12(b) for the example of eight bits being transferred from a computer to a printer. To transfer eight bits in parallel, it takes one time interval compared to eight time intervals for the serial transfer.

To summarize, an advantage of serial transfer of binary data is that a minimum of only one line is required. In parallel transfer, a number of lines equal to the number of bits to be transferred at one time is required. A disadvantage of serial transfer is that it takes longer to transfer a given number of bits than with parallel transfer. For example, if one bit can be transferred in 1 μs, then it takes 8 μs to serially transfer eight bits but only 1 μs to parallel transfer eight bits. A disadvantage of parallel transfer is that it takes more lines.

EXAMPLE 1–2

(a) Determine the total time required to serially transfer the eight bits contained in waveform *A* of Figure 1–13, and indicate the sequence of bits. The left-most bit is the first to be transferred. The 100 kHz clock is used as reference.

(b) What is the total time to transfer the same eight bits in parallel?



▲ **FIGURE 1–13**

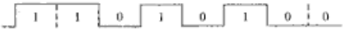
Solution **(a)** Since the frequency of the clock is 100 kHz, the period is

$$T = \frac{1}{f} = \frac{1}{100\text{ kHz}} = 10\ \mu\text{s}$$

It takes 10 μs to transfer each bit in the waveform. The total transfer time for 8 bits is

$$8 \times 10\ \mu s = \mathbf{80\ \mu s}$$

To determine the sequence of bits, examine the waveform in Figure 1–13 during each bit time. If waveform *A* is HIGH during the bit time, a 1 is transferred. If waveform *A* is LOW during the bit time, a 0 is transferred. The bit sequence is illustrated in Figure 1–14. The left-most bit is the first to be transferred.



▲ **FIGURE 1–14**

(b) A parallel transfer would take **10 μs** for all eight bits.

Supplementary Problem If binary data are transferred at the rate of 10 million bits per second (10 Mbits/s), how long will it take to parallel transfer 16 bits on 16 lines? How long will it take to serially transfer 16 bits?

SECTION 1–2 REVIEW

1. Define *binary*.
2. What does a *bit* mean?
3. What are the bits in a binary system?
4. How are the rise time and fall time of a pulse measured?
5. Knowing the period of a waveform, how do you find the frequency?
6. Explain what a clock waveform is.
7. What is the purpose of a timing diagram?
8. What is the main advantage of parallel transfer over serial transfer of binary data?

1–3 INTRODUCTION TO BASIC LOGIC OPERATIONS

In its basic form, logic is the realm of human reasoning that tells you a certain proposition (declarative statement) is true if certain conditions are true. Propositions can be classified as true or false. Many situations and processes that you encounter in your daily life can be expressed in the form of propositional, or logic, functions. Since such functions are true/false or yes/no statements, digital circuits with their two-state characteristics are applicable.

After completing this section, you should be able to

- List three basic logic operations
- Define the NOT operation
- Define the AND operation
- Define the OR operation

Several propositions, when combined, form propositional, or logic, functions. For example, the propositional statement "The light is on" will be true if "The bulb is not burned out" is true and if "The switch is on" is true. Therefore, this logical statement can be made: *The light is on only if the bulb is not burned out and the switch is on*. In this example, the first statement is true only if the last two statements are true. The first statement ("The light is on") is then the basic proposition, and the other two statements are the conditions on which the proposition depends.

In the 1850s, the Irish logician and mathematician George Boole developed a mathematical system for formulating logic statements with symbols so that problems can be written and solved in a manner similar to ordinary algebra. Boolean algebra, as it is known today, is applied in the design and analysis of digital systems and will be covered in detail in Chapter 4.

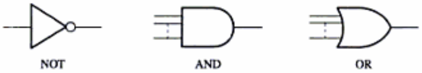
The term **logic** is applied to digital circuits used to implement logic functions. Several kinds of digital logic **circuits** are the basic elements that form the building blocks for such complex digital systems as the computer. We will now look at these elements and discuss their functions in a very general way. Later chapters will cover these circuits in detail.

Three basic logic operations (NOT, AND, and OR) are indicated by standard distinctive shape symbols in Figure 1–15. Other standard symbols for these logic operations will be introduced in Chapter 3. The lines connected to each symbol are the **inputs** and **outputs**. The inputs are on the left of each symbol and the output is on the right. A circuit that performs a specified logic operation (AND, OR) is called a logic **gate**. AND and OR gates can have any number of inputs, as indicated by the dashes in the figure.

In logic operations, the true/false conditions mentioned earlier are represented by a HIGH (true) and a LOW (false). Each of the three basic logic operations produces a unique response to a given set of conditions.

FIGURE 1–15

The basic logic operations and symbols

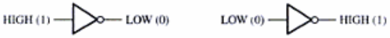


NOT

The **NOT** operation changes one logic level to the opposite logic level, as indicated in Figure 1–16. When the input is HIGH (1), the output is LOW (0). When the input is LOW, the output is HIGH. In either case, the output is *not* the same as the input. The NOT operation is implemented by a logic circuit known as an **inverter**.

FIGURE 1–16

The NOT operation

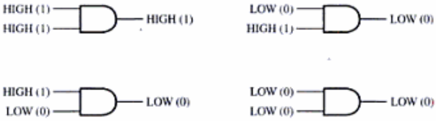

HIGH (1)
LOW (0)
LOW (0)
HIGH (1)

AND

The **AND** operation produces a HIGH output only if all the inputs are HIGH, as indicated in Figure 1–17 for the case of two inputs. When one input is HIGH *and* the other input is HIGH, the output is HIGH. When any or all inputs are LOW, the output is LOW. The AND operation is implemented by a logic circuit known as an *AND gate*.

FIGURE 1–17

The AND operation

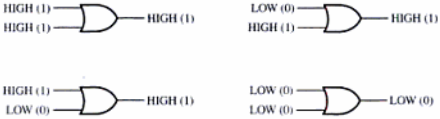

HIGH (1)
HIGH (1)
HIGH (1)
LOW (0)
HIGH (1)
LOW (0)
HIGH (1)
LOW (0)
LOW (0)
LOW (0)
LOW (0)
LOW (0)

OR

The **OR** operation produces a HIGH output when any of the inputs is HIGH, as indicated in Figure 1–18 for the case of two inputs. When one input is HIGH *or* the other input is HIGH *or* both inputs are HIGH, the output is HIGH. When both inputs are LOW, the output is LOW. The OR operation is implemented by a logic circuit known as an *OR gate*.

FIGURE 1–18

The OR operation


HIGH (1)
HIGH (1)
HIGH (1)
LOW (0)
HIGH (1)
HIGH (1)
HIGH (1)
LOW (0)
HIGH (1)
LOW (0)
LOW (0)
LOW (0)

SECTION 1–3 REVIEW

1. When does the NOT operation produce a HIGH output?
2. When does the AND operation produce a HIGH output?
3. When does the OR operation produce a HIGH output?
4. What is an inverter?
5. What is a logic gate?

1–4 DIGITAL INTEGRATED CIRCUITS

The basic logic operations that have been discussed and many more which will be discussed in later chapters are available in integrated circuit (IC) form. Digital systems have incorporated ICs for long because of their small size, high reliability, low cost, and low power consumption. It is important to be able to recognize the IC packages and to know how the pin connections are numbered.

After completing this section, you should be able to

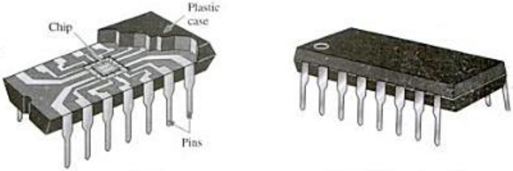
- Identify dual in-line packages (DIP)
- Determine pin number on DIP IC package

A monolithic integrated **circuit (IC)** is an electronic circuit that is constructed entirely on a single small chip of silicon. All the components that make up the circuit—transistors, diodes, resistors, and capacitors—are an integral part of that single chip. Fixed-function logic and programmable logic are two broad categories of digital ICs. In fixed-function logic, the logic functions are set by the manufacturer and cannot be altered.

IC Packages

Integrated circuit (IC) packages are classified according to the way they are mounted on printed circuit (PC) boards as either through-hole mounted or surface mounted. The through-hole type packages have pins (leads) that are inserted through holes in the PC board and can be soldered to conductors on the opposite side. The most common type of through-hole package is the dual in-line package (**DIP**) shown in Figure 1–19.

FIGURE 1–19



(a) Cutway view of DIP

(b) Dual in-line package (DIP)

Figure 1.19(a) shows a cutaway view with the circuit chip shown within the package. Points on the chip are connected to the package pins to allow input and output connections to the outside world. Figure 1.19(b) shows the outer view of DIP IC.

Pin Numbering

All IC packages have a standard format for numbering the pins (leads). The dual in-line package numbering arrangement is illustrated in Figure 1.20 for a 16-pin package. Looking at the top of the package, pin 1 is indicated by an identifier that can be either a small dot, a notch, or a bevelled edge. The dot is always next to pin 1. Also, with the notch oriented upward, pin 1 is always the top left pin, as indicated. Starting with pin 1, the pin numbers increase as you go down, then across and up. The highest pin number is always to the right of the notch or opposite the dot.

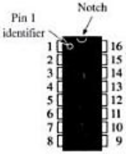


FIGURE 1–20

Pin numbering for DIP.

SECTION 1–4 REVIEW

1. What is an integrated circuit?
2. Define the terms DIP.

SUMMARY

- An analog quantity has continuous values.
- A digital quantity has a discrete set of values.
- A binary digit is called a bit.
- A pulse is characterized by rise time, fall time, pulse width, and amplitude.
- The frequency of a periodic waveform is the reciprocal of the period. The formulas relating frequency and period are

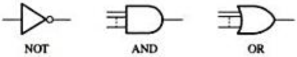
$$f = \frac{1}{T} \text{ and } T = \frac{1}{f}$$

- The duty cycle of a pulse waveform is the ratio of the pulse width to the period, expressed by the following formula as a percentage:

$$\text{Duty cycle} = \left(\frac{t_W}{T}\right)100\%$$

- A timing diagram is an arrangement of two or more waveforms showing their relationship with respect to time.
- Three basic logic operations are NOT, AND, and OR. The standard symbols for these are given in Figure 1–21.

▶ FIGURE 1–21



- Digital systems incorporate integrated circuits (IC). DIP is the most commonly used IC package for fixed-function ICs.

SELF-TEST

Answers are at the end of the chapter.

1. A quantity having continuous values is
   (a) a digital quantity (b) an analog quantity
   (c) a binary quantity (d) a natural quantity
2. The term *bit* means
   (a) a small amount of data (b) a 1 or a 0
   (c) binary digit (d) both answers (b) and (c)
3. The time interval on the leading edge of a pulse between 10% and 90% of the amplitude is the
   (a) rise time (b) fall time (c) pulse width (d) period

4. A pulse in a certain waveform occurs every 10 ms. The frequency is
   (a) 1 kHz **(b)** 1 Hz **(c)** 100 Hz **(d)** 10 Hz
5. In a certain digital waveform, the period is twice the pulse width. The duty cycle is
   (a) 100% **(b)** 200% **(c)** 50%
6. An inverter
   (a) performs the NOT operation **(b)** changes a HIGH to a LOW
   (c) changes a LOW to a HIGH **(d)** does all of the above
7. The output of an AND gate is HIGH when
   (a) any input is HIGH **(b)** all inputs are HIGH
   (c) no inputs are HIGH **(d)** both answers (a) and (b)
8. The output of an OR gate is HIGH when
   (a) any input is HIGH **(b)** all inputs are HIGH
   (c) no inputs are HIGH **(d)** both answers (a) and (b)

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

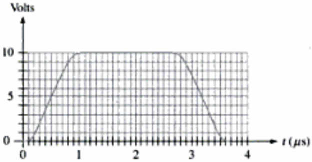
SECTION 1-1 Digital and Analog Quantities

1. Name two advantages of digital data as compared to analog data.
2. Name an analog quantity other than temperature and sound.

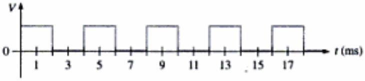
SECTION 1-2 Binary Digits, Logic Levels, and Digital Waveforms

3. Define the sequence of bits (1s and 0s) represented by each of the following sequences of levels:
   (a) HIGH, HIGH, LOW, HIGH, LOW, LOW, LOW, HIGH
   (b) LOW, LOW, LOW, HIGH, LOW, HIGH, LOW, HIGH, LOW
4. List the sequence of levels (HIGH and LOW) that represent each of the following bit sequences:
   (a) 1 0 1 1 1 0 1 **(b)** 1 1 1 0 1 0 0 1
5. For the pulse shown in Figure 1–22, graphically determine the following:
   (a) rise time **(b)** fall time **(c)** pulse width **(d)** amplitude

▶ FIGURE 1–22

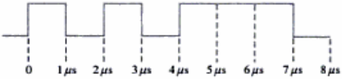


6. Determine the period of the digital waveform in Figure 1–23.
7. What is the frequency of the waveform in Figure 1–23?
8. Is the pulse waveform in Figure 1–23 periodic or nonperiodic?
9. Determine the duty cycle of the waveform in Figure 1–23.



▲ **FIGURE 1–23**

10. Determine the bit sequence represented by the waveform in Figure 1–24. A bit time is 1 μs in this case.
11. What is the total serial transfer time for the eight bits in Figure 1–24? What is the total parallel transfer time?



▲ **FIGURE 1–24**

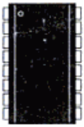
SECTION 1-3 Introduction to Basic Logic Operations

12. A logic circuit requires HIGHs on all its inputs to make the output HIGH. What type of logic circuit is it?
13. A basic 2-input logic circuit has a HIGH on one input and a LOW on the other input, and the output is LOW. Identify the circuit.
14. A basic 2-input logic circuit has a HIGH on one input and a LOW on the other input, and the output is HIGH. What type of logic circuit is it?

SECTION 1-4 Digital Integrated Circuits

15. What is meant by an IC?
16. Explain the term DIP.
17. Label the pin numbers on the IC package whose top view is shown in Figure 1–25.

▶ **FIGURE 1–25**



ANSWERS

SECTION REVIEWS

SECTION 1-1 Digital and Analog Quantities

1. *Analog* means continuous.
2. *Digital* means discrete.
3. A digital quantity has a discrete set of values and an analog quantity has continuous values.
4. A public address system is analog. A CD player is analog and digital. A computer is all digital.

SECTION 1-2 Binary Digits, Logic Levels, and Digital Waveforms

1. Binary means having two states or values.
2. A bit is a binary digit.
3. The bits are 1 and 0.
4. Rise time: from 10% to 90% of amplitude. Fall time: from 90% to 10% of amplitude.
5. Frequency is the reciprocal of the period.
6. A clock waveform is a basic timing waveform from which other waveforms are derived.
7. A timing diagram shows time relationships of waveforms.
8. Parallel transfer is faster than serial transfer.

SECTION 1-3 Introduction to Basic Logic Operations

1. When the input is LOW
2. When all inputs are HIGH
3. When any or all inputs are HIGH
4. An inverter is a NOT circuit.
5. A logic gate is a circuit that performs a logic operation (AND, OR).

SECTION 1-4 Digital Integrated Circuits

1. An integrated circuit (IC) is an electronic circuit constructed entirely on a semiconductor chip.
2. Dual in-line package.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

1–1 $f = 6.67$ kHz; Duty cycle = 16.7%

1–2 Parallel transfer: 100 ns; Serial transfer: 1.6 μs

SELF-TEST

1. (b) **2.** (d) **3.** (a) **4.** (c) **5.** (c) **6.** (d) **7.** (b) **8.** (d)

2

NUMBER SYSTEMS, OPERATIONS, AND CODES

CHAPTER OBJECTIVES

- Review the decimal number system
- Count in the binary number system
- Convert from decimal to binary and from binary to decimal
- Apply arithmetic operations to binary numbers
- Determine the 1's and 2's complements of a binary number
- Express signed binary numbers in sign-magnitude, 1's complement, 2's complement, and floating-point format
- Carry out arithmetic operations with signed binary numbers
- Convert between the binary and hexadecimal number systems
- Add numbers in hexadecimal form
- Convert between the binary and octal number systems
- Express decimal numbers in binary coded decimal (BCD) form
- Add BCD numbers
- Convert between the binary system and the Gray code
- Interpret the American Standard Code for Information Interchange (ASCII)
- Concept of parity and parity method of error detection
- Review of error-detection codes
- Hamming error-detection and error-correction

INTRODUCTION

The binary number system and digital codes are fundamental to computers and to digital electronics in general. In this chapter, the binary number system and its relationship to other number systems such as decimal, hexadecimal, and octal is the principal focus. Arithmetic operations with binary numbers are covered to provide a basis for understanding how computers and many other types of digital systems work. Also, digital codes such as binary coded decimal (BCD), the Gray code, and the ASCII are covered. The concept of parity and the parity method of detecting errors in codes is introduced. Various error detection codes and Hamming code for error correction has been covered.

2–1 DECIMAL NUMBERS

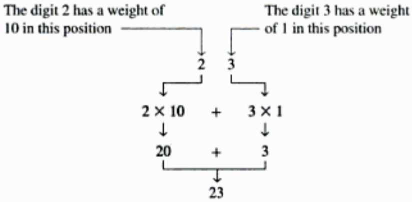
You are familiar with the decimal number system because you use decimal numbers every day. Although decimal numbers are commonplace, their weighted structure is often not understood. In this section, the structure of decimal numbers is reviewed. This review will help you more easily understand the structure of the binary number system, which is important in computers and digital electronics.

After completing this section, you should be able to

- Explain why the decimal number system is a weighted system
- Explain how powers of ten are used in the decimal system
- Determine the weight of each digit in a decimal number

The **decimal** number system has ten digits, 0 through 9. Each of the ten digits represents a certain quality. Since there are ten distinct digits used in the decimal number system, therefore, the decimal number system has a base (or radix) of 10.

As you know, the ten symbols (**digits**) do not limit you to expressing only ten different quantities because you use the various digits in appropriate positions within a number to indicate the magnitude of the quantity. You can express quantities up through nine before running out of digits; if you wish to express a quantity greater than nine, you use two or more digits, and the position of each digit within the number tells you the magnitude it represents. If, for example, you wish to express the quantity twenty-three, you use (by their respective positions in the number) the digit 2 to represent the quantity twenty and the digit 3 to represent the quantity three, as illustrated below:


The digit 2 has a weight of 10 in this position
The digit 3 has a weight of 1 in this position
2 3
2 × 10 + 3 × 1
20 + 3
23

The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a **weight**. The weights for whole numbers are positive powers of ten that increase from right to left, beginning with $10^0 = 1$.

$$\ldots 10^5 \ 10^4 \ 10^3 \ 10^2 \ 10^1 \ 10^0$$

For fractional numbers, the weights are negative powers of ten that decrease from left to right beginning with 10^{-1}.

$$10^2 \ 10^1 \ 10^0.10^{-1} \ 10^{-2} \ 10^{-3} \ldots$$

Decimal point

The value of a decimal number is the sum of the digits after each digit has been multiplied by its weight, as Examples 2–1 and 2–2 illustrate.

EXAMPLE 2–1

Express the decimal number 47 as a sum of the values of each digit.

Solution The digit 4 has a weight of 10, which is 10^1, as indicated by its position. The digit 7 has a weight of 1, which is 10^0, as indicated by its position.

$$\begin{aligned} 47 &= (4 \times 10^1) + (7 \times 10^0) \\ &= (4 \times 10) + (7 \times 1) = \mathbf{40 + 7} \end{aligned}$$

Supplementary Problem Determine the value of each digit in 939.

EXAMPLE 2–2

Express the decimal number 568.23 as a sum of the values of each digit.

Solution The whole number digit 5 has a weight of 100, which is 10^2, the digit 6 has a weight of 10, which is 10^1, the digit 8 has a weight of 1, which is 10^0, the fractional digit 2 has a weight of 0.1 is 10^{-1}, and the fractional digit 3 has a weight of 0.01, which is 10^{-2}.

$$\begin{aligned}568.23 &= (5 \times 10^2) + (6 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2})\\ &= (5 \times 100) + (6 \times 10) + (8 \times 1) + (2 \times 0.1) + (3 \times 0.01)\\ &= \mathbf{500} + \mathbf{60} + \mathbf{8} + \mathbf{0.2} + \mathbf{0.03}\end{aligned}$$

Supplementary Problem Determine the value of each digit in 67.924.

SECTION 2–1 REVIEW

Answers are at the end of the chapter.

1. What weight does the digit 7 have in each of the following numbers?
   (a) 1370 (b) 6725 (c) 7051 (d) 58.72
2. Express each of the following decimal numbers as a sum of the products obtained by multiplying each digit by its appropriate weight:
   (a) 51 (b) 137 (c) 1492 (d) 106.58

2–2 BINARY NUMBERS

The binary number system is simply another way to represent quantities. The binary system is less complicated than the decimal system because it has only two digits. It may seem more difficult at first because it is unfamiliar to you. The decimal system with its ten digits is a base-ten system; the binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and 0. The position of a 1 or 0 in a binary number indicates its weight, or value within the number, just as the position of a decimal digit determines the value of that digit. The weights in a binary number are based on powers of two.

After completing this section, you should be able to

▪ Count in binary ▪ Determine the largest decimal number that can be represented by a given number of bits ▪ Convert a binary number to a decimal number

Counting in Binary

To learn to count in the **binary** system, first look at how you count in the decimal system. You start at zero and count up to nine before you run out of digits. You then start another digit position (to the left) and continue counting 10 through 99. At this point you have exhausted all two-digit combinations, so a third digit position is needed to count from 100 through 999.

A comparable situation occurs when you count in binary, except that you have only two digits, called **bits**. Begin counting: 0, 1. At this point you have used both digits, so include another digit position and continue: 10, 11. You have now exhausted all combinations of two digits, so a third position is required. With three digit positions you can continue to count: 100, 101, 110, and 111. Now, you need a fourth digit position to continue, and so on. A binary count of zero through fifteen is shown in Table 2–1. Notice the patterns with which the 1s and 0s alternate in each column.

▶ TABLE 2–1

DECIMAL NUMBER	BINARY NUMBER			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

As you have seen in Table 2–1, four bits are required to count from zero to 15. The value of a bit is determined by its position in the number. In general, with n bits you can count up to a number equal to $2^n - 1$.

$$\text{Largest decimal number} = 2^n - 1$$

For example, with five bits ($n = 5$) you can count from zero to thirty-one.

$$2^5 - 1 = 32 - 1 = 31$$

With six bits ($n = 6$) you can count from zero to sixty-three.

$$2^6 - 1 = 64 - 1 = 63$$

The Weighting Structure of Binary Numbers

A binary number is a weighted number. The right-most bit is the **LSB** (least significant bit) in a binary whole number and has a weight of $2^0 = 1$. The weights increase from right to left by a power of two for each bit. The left-most bit is the **MSB** (most significant bit); its weight depends on the size of the binary number.

Fractional numbers can also be represented in binary by placing bits to the right of the binary point, just as fractional decimal digits are placed to the right of the decimal point. The left-most bit is the MSB in a binary fractional number and has a weight of $2^{-1} = 0.5$. The fractional weights decrease from left to right by a negative power of two for each bit.

The weight structure of a binary number is

$$2^{n-1} \ldots 2^3\, 2^2\, 2^1\, 2^0 \underset{\uparrow}{.}\, 2^{-1}\, 2^{-2} \ldots 2^{-n}$$

Binary point

where n is the number of bits from the binary point. Thus, all the bits to the left of the binary point have weights that are positive powers of two, as previously discussed for whole numbers. All bits to the right of the binary point have weights that are negative powers of two, or fractional weights.

The powers of two and their equivalent decimal weights for an 8-bit binary whole number and a 6-bit binary fractional number are shown in Table 2–2. Notice that the weight doubles for each positive power of two and that the weight is halved for each negative power of two. You can easily extend the table by doubling the weight of the most significant positive power of two and halving the weight of the least significant negative power of two; for example, $2^9 = 512$ and $2^{-7} = 0.0078125$.

▼ **TABLE 2–2**

Binary weights

POSITIVE POWERS OF TWO (WHOLE NUMBERS)									NEGATIVE POWERS OF TWO (FRACTIONAL NUMBER)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64
									0.5	0.25	0.125	0.0625	0.03125	0.015625

Binary-to-Decimal Conversion

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0. The following two examples will illustrate this.

EXAMPLE 2–3

Convert the binary whole number 1101101 to decimal.

Solution Determine the weight of each bit that is a 1, and then find the sum of the weights to get the decimal number.

$$\begin{array}{rccccccc} \text{Weight:} & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \text{Binary number:} & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

$$\begin{aligned} 1101101 &= 2^6 + 2^5 + 2^3 + 2^2 + 2^0 \\ &= 64 + 32 + 8 + 4 + 1 = \mathbf{109} \end{aligned}$$

Supplementary Problem Convert the binary number 10010001 to decimal.

EXAMPLE 2–4

Convert the fractional binary number 0.1011 to decimal.

Solution Determine the weight of each bit that is a 1, and then sum the weights to get the decimal fraction.

$$\begin{array}{rcccccc} \text{Weight:} & & & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\ \text{Binary number:} & 0 & . & 1 & 0 & 1 & 1 \end{array}$$

$$\begin{aligned} 0.1011 &= 2^{-1} + 2^{-3} + 2^{-4} \\ &= 0.5 + 0.125 + 0.0625 = \mathbf{0.6875} \end{aligned}$$

Supplementary Problem Convert the binary number 10.111 to decimal.

SECTION 2-2 REVIEW

1. What is the largest decimal number that can be represented in binary with eight bits?
2. Determine the weight of the 1 in the binary number 10000.
3. Convert the binary number 10111101.011 to decimal.

2-3 DECIMAL-TO-BINARY CONVERSION

In Section 2–2 you learned how to convert a binary number to the equivalent decimal number. Now, you will learn two ways of converting from a decimal number to a binary number.

After completing this section, you should be able to

- Convert a decimal number to binary using the sum-of-weights method
- Convert a decimal whole number to binary using the repeated division-by-2 method
- Convert a decimal fraction to binary using the repeated multiplication-by-2 method

Sum-of-Weights Method

One way to find the binary number that is equivalent to a given decimal number is to determine the set of binary weights whose sum is equal to the decimal number. An easy way to remember binary weights is that the lowest is 1, which is 2^0, and that by doubling any weight, you get the next higher weight; thus, a list of seven binary weights would be 64, 32, 16, 8, 4, 2, 1 as you learned in the last section. The decimal number 9, for example, can be expressed as the sum of binary weights as follows:

$$9 = 8 + 1 \quad \text{or} \quad 9 = 2^3 + 2^0$$

Placing 1s in the appropriate weight positions, 2^3 and 2^0, and 0s in the 2^2 and 2^1 positions determines the binary number for decimal 9.

$$\begin{array}{cccc} 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 0 & 1 \end{array} \quad \text{Binary number for decimal 9}$$

EXAMPLE 2-5

Convert the following decimal numbers to binary:

(a) 12 **(b)** 25 **(c)** 58 **(d)** 82

Solution

(a) $12 = 8 + 4 = 2^3 + 2^2 \longrightarrow$ **1100**

(b) $25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 \longrightarrow$ **11001**

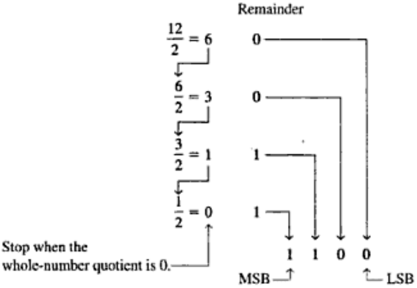
(c) $58 = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 \longrightarrow$ **111010**

(d) $82 = 64 + 16 + 2 = 2^6 + 2^4 + 2^1 \longrightarrow$ **1010010**

Supplementary Problem Convert the decimal number 125 to binary.

Repeated Division-by-2 Method

A systematic method of converting whole numbers from decimal to binary is the *repeated division-by-2* process. For example, to convert the decimal number 12 to binary, begin by dividing 12 by 2. Then, divide each resulting quotient by 2 until there is a 0 whole-number quotient. The **remainders** generated by each division form the binary number. The first remainder to be produced is the LSB (least significant bit) in the binary number, and the last remainder to be produced is the MSB (most significant bit). This procedure is shown in the following steps for converting the decimal number 12 to binary.

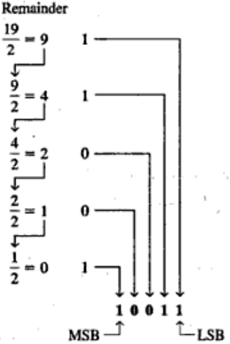

Remainder
12/2 = 6 0
6/2 = 3 0
3/2 = 1 1
1/2 = 0 1
Stop when the whole-number quotient is 0.
1 1 0 0
MSB
LSB

EXAMPLE 2–6

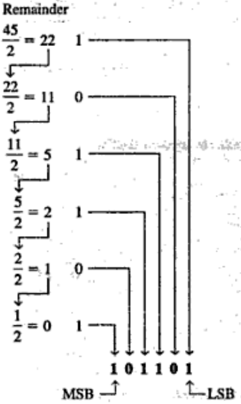
Convert the following decimal numbers to binary:

(a) 19 **(b)** 45

Solution **(a)**


Remainder
19/2 = 9 1
9/2 = 4 1
4/2 = 2 0
2/2 = 1 0
1/2 = 0 1
1 0 0 1 1
MSB
LSB

(b)


Remainder
45/2 = 22 1
22/2 = 11 0
11/2 = 5 1
5/2 = 2 1
2/2 = 1 0
1/2 = 0 1
1 0 1 1 0 1
MSB
LSB

Supplementary Problem Convert decimal number 39 to binary.

Converting Decimal Fractions to Binary

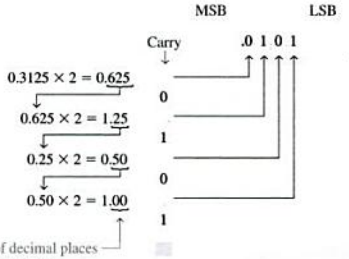
Examples 2–5 and 2–6 demonstrated whole-number conversions. Now, let us look at fractional conversions. An easy way to remember fractional binary weights is that the most significant weight is 0.5, which is 2^{-1}, and that by halving any weight, you get the next lower weight; thus a list of four fractional binary weights would be 0.5, 0.25, 0.125, 0.0625.

Sum-of-Weights The sum-of-weights method can be applied to fractional decimal numbers, as shown in the following example:

$$0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$$

There is a 1 in the 2^{-1} position, a 0 in the 2^{-2} position, and a 1 in the 2^{-3} position.

Repeated Multiplication by 2 As you have seen, decimal whole numbers can be converted to binary by repeated division by 2. Decimal fractions can be converted to binary by repeated multiplication by 2. For example, to convert the decimal fraction 0.3125 to binary, begin by multiplying 0.3125 by 2 and then multiplying each resulting fractional part of the product by 2 until the fractional product is zero or until the desired number of decimal places is reached. The carried digits, or **carries**, generated by the multiplications produce the binary number. The first carry produced is the MSB, and the last carry is the LSB. This procedure is illustrated as follows:



Continue to the desired number of decimal places or stop when the fractional part is all zeros.

SECTION 2–3 REVIEW

1. Convert each decimal number to binary by using the sum-of-weights method:
   (a) 23 (b) 57 (c) 45.5
2. Convert each decimal number to binary by using the repeated division-by-2 method (repeated multiplication-by-2 for fractions):
   (a) 14 (b) 21 (c) 0.375

2–4 BINARY ARITHMETIC

Binary arithmetic is essential in all digital computers and in many other types of digital systems. To understand digital systems, you must know the basics of binary addition, subtraction, multiplication, and division. This section provides an introduction that will be expanded in later sections.

After completing this section, you should be able to

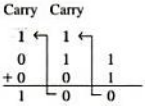
- Add binary numbers
- Subtract binary numbers
- Multiply binary numbers
- Divide binary numbers

Binary Addition

The four basic rules for adding binary digits (bits) are as follows:

0 + 0 = 0	Sum of 0 with a carry of 0
0 + 1 = 1	Sum of 1 with a carry of 0
1 + 0 = 1	Sum of 1 with a carry of 0
1 + 1 = 10	Sum of 0 with a carry of 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two 1s yields a binary two (10). When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following addition of 11 + 1:



In the right column, 1 + 1 = 0 with a carry of 1 to the next column to the left. In the middle column, 1 + 1 + 0 = 0 with a carry of 1 to the next column to the left. In the left column, 1 + 0 + 0 = 1.

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit). This situation is illustrated as follows:

Carry bits ↓

1 + 0 + 0 = 01	Sum of 1 with a carry of 0
1 + 1 + 0 = 10	Sum of 0 with a carry of 1
1 + 0 + 1 = 10	Sum of 0 with a carry of 1
1 + 1 + 1 = 11	Sum of 1 with a carry of 1

Example 2–7 illustrates binary addition.

EXAMPLE 2–7

Add the following binary numbers:

(a) 11 + 11 **(b)** 100 + 10 **(c)** 111 + 11 **(d)** 110 + 100

Solution The equivalent decimal addition is also shown for reference.

$$\textbf{(a)}\ \begin{array}{r} 11 \\ \underline{+11} \\ \mathbf{110} \end{array} \quad \begin{array}{r} 3 \\ \underline{+3} \\ 6 \end{array} \qquad \textbf{(b)}\ \begin{array}{r} 100 \\ \underline{+10} \\ \mathbf{110} \end{array} \quad \begin{array}{r} 4 \\ \underline{+2} \\ 6 \end{array} \qquad \textbf{(c)}\ \begin{array}{r} 111 \\ \underline{+11} \\ \mathbf{1010} \end{array} \quad \begin{array}{r} 7 \\ \underline{+3} \\ 10 \end{array} \qquad \textbf{(d)}\ \begin{array}{r} 110 \\ \underline{+100} \\ \mathbf{1010} \end{array} \quad \begin{array}{r} 6 \\ \underline{+4} \\ 10 \end{array}$$

Supplementary Problem Add 1111 and 1100.

Binary Subtraction

The four basic rules for subtracting bits are as follows:

0 − 0 = 0	
1 − 1 = 0	
1 − 0 = 1	
10 − 1 = 1	0 − 1 with a borrow of 1

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied. Examples 2–8 and 2–9 illustrate binary subtraction; the equivalent decimal subtractions are also shown.

EXAMPLE 2–8

Perform the following binary subtractions:

(a) 11 − 01 **(b)** 11 − 10

Solution **(a)**

$$\begin{array}{r} 11 \\ -01 \\ \hline \mathbf{10} \end{array} \qquad \begin{array}{r} 3 \\ -1 \\ \hline 2 \end{array}$$

(b)

$$\begin{array}{r} 11 \\ -10 \\ \hline \mathbf{01} \end{array} \qquad \begin{array}{r} 3 \\ -2 \\ \hline 1 \end{array}$$

No borrows were required in this example. The binary number 01 is the same as 1.

Supplementary Problem Subtract 100 from 111.

EXAMPLE 2–9

Subtract 011 from 101.

Solution

$$\begin{array}{r} 101 \\ -011 \\ \hline \mathbf{010} \end{array} \qquad \begin{array}{r} 5 \\ -3 \\ \hline 2 \end{array}$$

Let us examine exactly what was done to subtract the two binary numbers since a borrow is required. Begin with the right column.

Left column:
When a 1 is borrowed,
a 0 is left, so 0 − 0 = 0.

Middle column:
Borrow 1 from next column
to the left, making a 10 in this
column, then 10 − 1 = 1.

$$\begin{array}{r} \overset{0}{\cancel{1}}\,{}^{1}0\,1 \\ -0\,1\,1 \\ \hline 0\,1\,0 \end{array}$$

Right column:
1 − 1 = 0

Supplementary Problem Subtract 101 from 110.

Binary Multiplication

The four basic rules for multiplying bits are as follows:

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Multiplication is performed with binary numbers in the same manner as with decimal numbers. It involves forming partial products, shifting each successive partial product left one place, and then adding all the partial products. Example 2–10 will illustrate the procedure; the equivalent decimal multiplications are shown for reference.

EXAMPLE 2–10

Perform the following binary multiplications:

(a) 11 × 11 **(b)** 101 × 111

Solution **(a)**

$$\begin{array}{rr} & 11 \\ & \underline{\times 11} \\ \text{Partial} & 11 \\ \text{products} & \underline{+11} \\ & \mathbf{1001} \end{array} \qquad \begin{array}{r} 3 \\ \underline{\times 3} \\ 9 \end{array}$$

(b)

$$\begin{array}{rr} & 111 \\ & \underline{\times 101} \\ & 111 \\ \text{Partial products} & 000 \\ & \underline{+111} \\ & \mathbf{100011} \end{array} \qquad \begin{array}{r} 7 \\ \underline{\times 5} \\ 35 \end{array}$$

Supplementary Problem Multiply 1101 × 1010.

Binary Division

Division in binary follows the same procedure as division in decimal, as Example 2–11 illustrates. The equivalent decimal divisions are also given.

EXAMPLE 2–11

Perform the following binary divisions:

(a) 110 ÷ 11 **(b)** 110 ÷ 10

Solution

(a)
$$\begin{array}{r} 10 \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array} \qquad \begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

(b)
$$\begin{array}{r} 11 \\ 10\overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

Supplementary Problem Divide 1100 by 100.

SECTION 2–4 REVIEW

1. Perform the following binary additions:
   (*a*) 1101 + 1010 (*b*) 10111 + 01101
2. Perform the following binary subtractions:
   (*a*) 1101 − 0100 (*b*) 1001 − 0111
3. Perform the indicated binary operations:
   (*a*) 110 × 111 (*b*) 1100 ÷ 011

2–5 1'S AND 2'S COMPLEMENTS OF BINARY NUMBERS

The 1's complement and the 2's complement of a binary number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

After completing this section, you should be able to

- Convert a binary number to its 1's complement
- Convert a binary number to its 2's complement using either of two methods

Finding the 1's Complement of a Binary Number

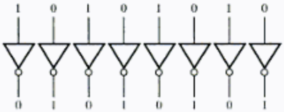
The 1's **complement** of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

1 0 1 1 0 0 1 0	Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 1 0 1	1's complement

Application Example

The simplest way to obtain the 1's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits), as shown in Figure 2–1 for an 8-bit binary number.

▶ FIGURE 2–1



Finding the 2's Complement of a Binary Number

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

2's complement = (1's complement) + 1

Example 2–12 shows how to find the 2's complement.

EXAMPLE 2–12

Find the 2's complement of 10110010.

Solution

10110010	Binary number
01001101	1's complement
+ 1	Add 1
01001110	2's complement

Supplementary Problem Determine the 2's complement of 11001011.

An alternative method of finding the 2's complement of a binary number is as follows:

1. Start at the right with the LSB and write the bits as they are up to and including the first 1.
2. Take the 1's complements of the remaining bits.

Example 2–13 illustrates these steps.

EXAMPLE 2–13

Find the 2's complement of 10111000 using the alternative method.

Solution

10111000 **Binary number**
1's complements of original bits → **01001000** 2's complement
↑ These bits stay the same.

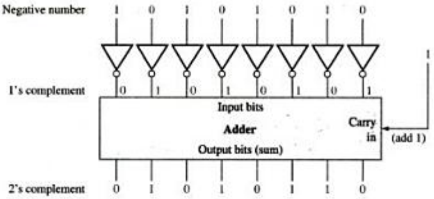
Supplementary Problem Find the 2's complement of 11000000.

Application Example

The 2's complement of a negative binary number can be realized using inverters and an adder, as indicated in Figure 2–2. This illustrates how an 8-bit number can be converted to its 2's complement by first inverting each bit (taking the 1's complement) and then adding 1 to the 1's complement with an adder circuit.

▶ FIGURE 2–2

Example of obtaining the 2's complement of a negative binary number



To convert from a 1's or 2's complement back to the true (uncomplemented) binary form, use the same two procedures described previously. To go from the 1's complement back to true binary, reverse all the bits. To go from the 2's complement form back to true binary, take the 1's complement of the 2's complement number and add 1 to the least significant bit.

SECTION 2–5 REVIEW

1. Determine the 1's complement of each binary number:
   (a) 00011010 (b) 11110111 (c) 10001101
2. Determine the 2's complement of each binary number:
   (a) 00010110 (b) 11111100 (c) 10010001

2–6 SIGNED NUMBERS

Digital systems, such as the computer, must be able to handle both positive and negative numbers. A signed binary number consists of both sign and magnitude information. The sign indicates whether a number is positive or negative and the magnitude is the value of the number. There are three forms in which signed integer (whole) numbers can be represented in binary: sign-magnitude, 1's complement, and 2's complement. Of these, the 2's complement is the most important and the sign-magnitude is rarely used. Noninteger and very large or small numbers can be expressed in floating-point format.

After completing this section, you should be able to

- Express positive and negative numbers in sign-magnitude
- Express positive and negative numbers in 1's complement
- Express positive and negative numbers in 2's complement
- Determine the decimal value of signed binary numbers
- Express a binary number in floating-point format

The Sign Bit

The left-most bit in a signed binary number is the **sign bit**, which tells you whether the number is positive or negative.

A 0 is for positive, and a 1 is for negative.

Sign-Magnitude Form

When a signed binary number is represented in sign-magnitude, the left-most bit is the sign bit and the remaining bits are the magnitude bits. The magnitude bits are in true (uncomplemented) binary for both positive and negative numbers. For example, the decimal number +25 is expressed as an 8-bit signed binary number using the sign-magnitude form as

$$\underset{\text{Sign bit}\uparrow}{0}\underbrace{0011001}_{\text{Magnitude bits}}$$

The decimal number −25 is expressed as

10011001

Notice that the only difference between +25 and −25 is the sign bit because the magnitude bits are in true binary for both positive and negative numbers.

In the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number but the sign bit is a 1 rather than a zero.

1's Complement Form

Positive numbers in 1's complement form are represented the same way as the positive sign-magnitude numbers. Negative numbers, however, are the 1's complements of the corresponding positive numbers. For example, using eight bits, the decimal number −25 is expressed as the 1's complement of +25 (00011001) as

11100110

In the 1's complement form, a negative number is the 1's complement of the corresponding positive number.

2's Complement Form

Positive numbers in 2's complement form are represented the same way as in the sign-magnitude and 1's complement forms. Negative numbers are the 2's complements of the corresponding positive numbers. Again, using eight bits, let us take decimal number −25 and express it as the 2's complement of +25 (00011001).

11100111

In the 2's complement form, a negative number is the 2's complement of the corresponding positive number.

EXAMPLE 2–14

Express the decimal number −39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

Solution First, write the 8-bit number for +39.

00100111

In the *sign-magnitude form*, −39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

10100111

In the *1's complement form*, −39 is produced by taking the 1's complement of +39 (00100111).

11011000

In the *2's complement form*, −39 is produced by taking the 2's complement of +39 (00100111) as follows:

$$\begin{array}{rl} 11011000 & \text{1's complement} \\ +\qquad\quad 1 & \\ \hline \mathbf{11011001} & \text{2's complement} \end{array}$$

Supplementary Problem Express +19 and −19 in sign-magnitude, 1's complement, and 2's complement.

The Decimal Value of Signed Numbers

Sign-magnitude Decimal values of positive and negative numbers in the sign-magnitude form are determined by summing the weights in all the magnitude bit positions where there are 1s and ignoring those positions where there are zeros. The sign is determined by examination of the sign bit. This is illustrated by Example 2–15.

EXAMPLE 2–15

Determine the decimal value of this signed binary number expressed in sign-magnitude: 10010101.

Solution The seven magnitude bits and their powers-of-two weights are as follows:

$$\begin{array}{ccccccc} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

Summing the weights where there are 1s,

$$16 + 4 + 1 = 21$$

The sign bit is 1; therefore, the decimal number is **−21**.

Supplementary Problem Determine the decimal value of the sign-magnitude number 01110111.

1's Complement Decimal values of positive numbers in the 1's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result. This is illustrated by Example 2–16.

EXAMPLE 2–16

Determine the decimal values of the signed binary numbers expressed in 1's complement:

(a) 00010111 **(b)** 11101000

Solution **(a)** The bits and their powers-of-two weights for the positive number are as follows:

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Summing the weights where there are 1s,

$$16 + 4 + 2 + 1 = \mathbf{+23}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of -2^7 or −128.

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Summing the weights where there are 1s,

$$-128 + 64 + 32 + 8 = -24$$

Adding 1 to the result, the final decimal number is

$$-24 + 1 = \mathbf{-23}$$

Supplementary Problem Determine the decimal value of the 1's complement number 11101011.

2's Complement Decimal values of positive and negative numbers in the 2's complement form are determined by summing the weights in all bit positions where there are 1s and ignoring those positions where there are zeros. The weight of the sign bit in a negative number is given a negative value. This is illustrated by Example 2–17.

EXAMPLE 2–17

Determine the decimal values of the signed binary numbers expressed in 2's complement:

(a) 01010110 **(b)** 10101010

Solution **(a)** The bits and their powers-of-two weights for the positive number are as follows:

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Summing the weights where there are 1s,

$$64 + 16 + 4 + 2 = \mathbf{+86}$$

(b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Summing the weights where there are 1s,

$$-128 + 32 + 8 + 2 = \mathbf{-86}$$

Supplementary Problem Determine the decimal value of the 2's complement number 11010111.

From these examples, you can see why the 2's complement form is preferred for representing signed integer numbers: To convert to decimal, it simply requires a summation of weights regardless of whether the number is positive or negative. The 1's complement system requires adding 1 to the summation of weights for negative numbers but not for positive numbers. Also, the 1's complement form is generally not used because two representations of zero (00000000 or 11111111) are possible.

Range of Signed Integer Numbers that can be Represented

The range of magnitude of a binary number depends on the number of bits (n). Here, we have used 8-bit numbers for illustration because the 8-bit grouping is common in most computers and has been given the special name **byte**. With one byte or eight bits, you can represent 256 different numbers. With two bytes or sixteen bits, you can represent 65,536 different numbers. With four bytes or 32 bits, you can represent 4.295×10^9 different numbers. The formula for finding the number of different combinations of n bits is

$$\text{Total combinations} = 2^n$$

For 2's complement signed numbers, the range of values for n-bit numbers is

$$-(2^{n-1}) \quad \text{to} \quad +(2^{n-1} - 1)$$

where in each case there is one sign bit and $n - 1$ magnitude bits. For example, with four bits you can represent numbers in 2's complement ranging from $-(2^3) = -8$ to $2^3 - 1 = +7$. Similarly, with eight bits you can go from -128 to $+127$, with sixteen bits you can go from $-32{,}768$ to $+32{,}767$, and so on.

Floating-Point Numbers

To represent very large **integer** (whole) numbers, many bits are required. There is also a problem when numbers with both integer and fractional parts, such as 23.5618, need to be represented. The floating-point number system, based on scientific notation, is capable of representing very large and very small numbers without an increase in the number of bits and also for representing numbers that have both integer and fractional components; it uses powers of ten.

A **floating-point number** (also known as a *real number*) consists of two parts plus a sign. The **mantissa** is the part of a floating-point number that represents the magnitude of the number. The **exponent** is the part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

A decimal example will be helpful in understanding the basic concept of floating-point numbers. Let us consider a decimal number which, in integer form, is 241,506,800. The mantissa is .2415068 and the exponent is 9. When the integer is expressed as a floating-point number, it is normalized by moving the decimal point to the left of all the digits so that the mantissa is a fractional number and the exponent is the power of ten. The floating-point number is written as

$$0.2415068 \times 10^9$$

For binary floating-point numbers, the format is defined by ANSI/IEEE Standard 754-1985 in three forms: *single-precision, double-precision*, and *extended-precision*. All these have the same basic formats except for the number of bits. Single-precision floating-point numbers have 32 bits, double-precision numbers have 64 bits, and extended-precision numbers have 80 bits. We will restrict our discussion to the single-precision floating-point format.

Single-Precision Floating-Point Binary Numbers In the standard format for a single-precision binary number, the sign bit (S) is the left-most bit, the exponent (E) includes the next eight bits, and the mantissa or fractional part (F) includes the remaining 23 bits, as shown next.

← 32 bits →

S	Exponent (E)	Mantissa (fraction, F)
1 bit	8 bits	23 bits

In the mantissa or fractional part, the binary point is understood to be to the left of the 23 bits. Effectively, there are 24 bits in the mantissa because in any binary number the left-most (most significant) bit is always a 1. Therefore, this 1 is understood to be there although it does not occupy an actual bit position.

The eight bits in the exponent represent a *biased exponent*, which is obtained by adding 127 to the actual exponent. The purpose of the bias is to allow very large or very small numbers without requiring a separate sign bit for the exponents. The biased exponent allows a range of actual exponent values from -126 to $+128$.

To illustrate how a binary number is expressed in floating-point format, let us use 1011010010001 as an example. First, it can be expressed as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiplying by the appropriate power of two.

$$1011010010001 = 1.011010010001 \times 2^{12}$$

Assuming that this is a positive number, the sign bit (S) is 0. The exponent, 12, is expressed as a biased exponent by adding it to 127 ($12 + 127 = 139$). The biased exponent (E) is expressed as the binary number 10001011. The mantissa is the fractional part (F) of the binary number, .011010010001. Because there is always a 1 to the left of the binary point in the power-of-two expression, it is not included in the mantissa. The complete floating-point number is

S	E	F
0	10001011	01101001000100000000000

Next, let us see how to evaluate a binary number that is already in floating-point format. The general approach to determining the value of a floating-point number is expressed by the following formula:

$$\text{Number} = (-1)^S(1 + F)(2^{E-127})$$

To illustrate, let us consider the following floating-point binary number:

S	E	F
1	10010001	10001110001000000000000

The sign bit is 1. The biased exponent is 10010001 = 145. Applying the formula, we get

$$\begin{aligned}\text{Number} &= (-1)^1(1.10001110001)(2^{145-127})\\ &= (-1)(1.10001110001)(2^{18}) = -1100011100010000000\end{aligned}$$

This floating-point binary number is equivalent to −407,688 in decimal. Since the exponent can be any number between −126 and +128, extremely large and small numbers can be expressed. A 32-bit floating-point number can replace a binary integer number having 129 bits. Because the exponent determines the position of the binary point, numbers containing both integer and fractional parts can be represented.

There are two exceptions to the format for floating-point numbers: The number 0.0 is represented by all 0s, and infinity is represented by all 1s in the exponent and all 0s in the mantissa.

EXAMPLE 2–18

Convert the decimal number 3.248×10^4 to a single-precision floating-point binary number.

Solution Convert the decimal number to binary.

$$3.248 \times 10^4 = 32480 = 111111011100000_2 = 1.11111011100000 \times 2^{14}$$

The MSB will not occupy a bit position because it is always a 1. Therefore, the mantissa is the fractional 23-bit binary number 11111011100000000000000 and the biased exponent is

$$14 + 127 = 141 = 10001101_2$$

The complete floating-point number is

0	**10001101**	**11111011100000000000000**

Supplementary Problem Determine the binary value of the following floating-point binary number:

0 10011000 10000100010100110000000

SECTION 2–6 REVIEW

1. Express the decimal number +9 as an 8-bit binary number in the sign-magnitude system.
2. Express the decimal number −33 as an 8-bit binary number in the 1's complement system.
3. Express the decimal number −46 as an 8-bit binary number in the 2's complement system.
4. List the two parts of a signed, floating-point number.

2–7 ARITHMETIC OPERATIONS WITH SIGNED NUMBERS

In the last section, you learned how signed numbers are represented in three different forms. In this section, you will learn how signed numbers are added, subtracted, multiplied, and divided. Because the 2's complement form for representing signed numbers is the most widely used in computers and microprocessor-based systems, the coverage in this section is limited to 2's complement arithmetic. The processes covered can be extended to the other forms if necessary.

After completing this section, you should be able to

- Add signed binary numbers
- Explain how computers add strings of numbers
- Define *overflow*
- Subtract signed binary numbers
- Multiply signed binary numbers using the direct addition method
- Multiply signed binary numbers using the partial products method
- Divide signed binary numbers

Addition

The two numbers in an addition are the **addend** and the **augend**. The result is the **sum**. There are four cases that can occur when two signed binary numbers are added.

1. Both numbers positive
2. Positive number with magnitude larger than negative number
3. Negative number with magnitude larger than positive number
4. Both numbers negative

Let us take one case at a time using 8-bit signed numbers as examples. The equivalent decimal numbers are shown for reference.

Both numbers positive:

$$\begin{array}{r} 00000111 \\ +\ 00000100 \\ \hline 00001011 \end{array} \qquad \begin{array}{r} 7 \\ +\ 4 \\ \hline 11 \end{array}$$

The sum is positive and is therefore in true (uncomplemented) binary.

Positive number with magnitude larger than negative number:

$$\begin{array}{rr} & 00001111 \\ & +\ 11111010 \\ \cline{2-2} \text{Discard carry} \longrightarrow 1 & 00001001 \end{array} \qquad \begin{array}{r} 15 \\ +\ -6 \\ \hline 9 \end{array}$$

The final carry bit is discarded. The sum is positive and therefore in true (uncomplemented) binary.

Negative number with magnitude larger than positive number:

$$\begin{array}{r} 00010000 \\ +\ 11101000 \\ \hline 11111000 \end{array} \qquad \begin{array}{r} 16 \\ +\ -24 \\ \hline -8 \end{array}$$

The sum is negative and therefore in 2's complement form.

Both numbers negative:

$$\begin{array}{rr} & 11111011 \\ & +\ 11110111 \\ \cline{2-2} \text{Discard carry} \longrightarrow 1 & 11110010 \end{array} \qquad \begin{array}{r} -5 \\ +\ -9 \\ \hline -14 \end{array}$$

The final carry bit is discarded. The sum is negative and therefore in 2's complement form.

The above discussions lead to the following conclusions:

1. Addition of two positive numbers yields a positive number.
2. Addition of a positive number and a smaller negative number yields a positive number.
3. Addition of a positive number and a larger negative yields a negative number in 2's complement.

In a computer, the negative numbers are stored in 2's complement form so, as you can see, the addition process is very simple: *Add the two numbers and discard any final carry bit.*

Overflow Condition When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an **overflow** results as indicated by an incorrect sign bit. An overflow can occur only when both numbers are positive or both numbers are negative. The following 8-bit example will illustrate this condition.

$$\begin{array}{rr} 01111101 & 126 \\ +\underline{00111010} & +\underline{58} \\ 1\underbrace{0110111} & 183 \end{array}$$

Sign incorrect

Magnitude incorrect

In this example the sum of 183 requires eight magnitude bits. Since there are seven magnitude bits in the numbers (one bit is the sign), there is a carry into the sign bit which produces the overflow indication.

Numbers are Added Two at a Time Now let us look at the addition of a string of numbers, added two at a time. This can be accomplished by adding the first two numbers, then adding the third number to the sum of the first two, then adding the fourth number to this result, and so on. This is how computers add strings of numbers. The addition of numbers taken two at a time is illustrated in Example 2–19.

EXAMPLE 2–19

Add the signed numbers: 01000100, 00011011, 00001110, and 00010010.

Solution The equivalent decimal additions are given for reference.

68	01000100	
+ 27	+ 00011011	Add 1st two numbers
95	01011111	1st sum
+ 14	+ 00001110	Add 3rd number
109	01101101	2nd sum
+ 18	+ 00010010	Add 4th number
127	**01111111**	Final sum

Supplementary Problem Add 00110011, 10111111, and 01100011. These are signed numbers.

Subtraction

Subtraction is a special case of addition. For example, subtracting +6 (the **subtrahend**) from +9 (the **minuend**) is equivalent to adding −6 to +9. Basically, *the subtraction operation changes the sign of the subtrahend and adds it to the minuend.* The result of a subtraction is called the **difference**.

The sign of a positive or negative binary number is changed by taking its 2's complement.

For example, when you take the 2's complement of the positive number 00000100 (+4), you get 11111100, which is −4 as the following sum-of-weights evaluation shows:

$$-128 + 64 + 32 + 16 + 8 + 4 = -4$$

As another example, when you take the 2's complement of the negative number 11101101 (−19), you get 00010011, which is +19 as the following sum-of-weights evaluation shows:

$$16 + 2 + 1 = 19$$

Since subtraction is simply an addition with the sign of the subtrahend changed, the process is stated as follows:

To subtract two signed numbers, take the 2's complement of the subtrahend and add. Discard any final carry bit.

Example 2–20 illustrates the subtraction process.

EXAMPLE 2–20

Perform each of the following subtractions of the signed numbers:

(a) 00001000 − 00000011 **(b)** 00001100 − 11110111

(c) 11100111 − 00010011 **(d)** 10001000 − 11100010

Solution Like in other examples, the equivalent decimal subtractions are given for reference.

(a) In this case, 8 − 3 = 8 + (−3) = 5.

$$\begin{array}{rrl} & 00001000 & \text{Minuend } (+8) \\ & \underline{+\ 11111101} & \text{2's complement of subtrahend } (-3) \\ \text{Discard carry} \longrightarrow & \mathbf{1\ 00000101} & \text{Difference } (+5) \end{array}$$

(b) In this case, 12 − (−9) = 12 + 9 = 21.

$$\begin{array}{rl} 00001100 & \text{Minuend } (+12) \\ \underline{+\ 00001001} & \text{2's complement of subtrahend } (+9) \\ 00010101 & \text{Difference } (+21) \end{array}$$

(c) In this case, −25 − (+19) = −25 + (−19) = −44.

$$\begin{array}{rrl} & 11100111 & \text{Minuend } (-25) \\ & \underline{+\ 11101101} & \text{2's complement of subtrahend } (-19) \\ \text{Discard carry} \longrightarrow & \mathbf{1\ 11010100} & \text{Difference } (-44) \end{array}$$

(d) In this case, −120 − (−30) = −120 + 30 = −90.

$$\begin{array}{rl} 10001000 & \text{Minuend } (-120) \\ \underline{+\ 00011110} & \text{2's complement of subtrahend } (+30) \\ \mathbf{10100110} & \text{Difference } (-90) \end{array}$$

Supplementary Problem Subtract 01000111 from 01011000.

Multiplication

The numbers in a multiplication are the **multiplicand**, the **multiplier**, and the **product**. These are illustrated in the following decimal multiplication:

8	Multiplicand
× 3	Multiplier
24	Product

Since multiplication is equivalent to adding a number to itself a number of times equal to the multiplier, therefore, the multiplication operation in most computers is accomplished using addition. As you have already seen, subtraction is done with an adder; now let us see how multiplication is done.

Direct addition and *partial products* are two basic methods for performing multiplication using addition. In the direct addition method, you add the multiplicand a number of times equal to the multiplier. In the previous decimal example (3 × 8), three multiplicands are added: 8 + 8 + 8 = 24. The disadvantage of this approach is that it becomes very lengthy if the multiplier is a large number. For example, to multiply 75 × 350, you must add 350 to itself 75 times. Incidentally, this is why the term *times* is used to mean multiply.

The partial products method is perhaps the more common one because it reflects the way you multiply longhand. The multiplicand is multiplied by each multiplier digit beginning with the least significant digit. The result of the multiplication of the multiplicand by a multiplier digit is called a *partial product*. Each successive partial product is moved (shifted) one place to the left and when all the partial products have been produced, they are added to get the final product. Here is a decimal example.

239	Multiplicand
× 123	Multiplier
717	1st partial product (3 × 239)
478	2nd partial product (2 × 239)
+ 239	3rd partial product (1 × 239)
29,397	Final product

The sign of the product of a multiplication depends on the signs of the multiplicand and the multiplier according to the following two rules:

- **If the signs are the same, the product is positive.**
- **If the signs are different, the product is negative.**

When two binary numbers are multiplied, both numbers must be in true (uncomplemented) form. The direct addition method is illustrated in Example 2–21 adding two binary numbers at a time.

EXAMPLE 2–21

Multiply the signed binary numbers: 01001101 (multiplicand) and 00000100 (multiplier) using the direct addition method.

Solution Since both numbers are positive, they are in true form, and the product will be positive. The decimal value of the multiplier is 4, so the multiplicand is added to itself four times as follows:

01001101	1st time
+ 01001101	2nd time
10011010	Partial sum
+ 01001101	3rd time
11100111	Partial sum
+ 01001101	4th time
100110100	Product

Since the sign bit of the multiplicand is 0, it has no effect on the outcome. All of the bits in the product are magnitude bits.

Supplementary Problem Multiply 01100001 by 00000110.

Now, let us look at the partial products method of binary multiplication. The basic steps in the process are as follows:

Step 1. Determine if the signs of the multiplicand and multiplier are the same or different. This determines what the sign of the product will be.

Step 2. Change any negative number to true (uncomplemented) form. Because most computers store negative numbers in 2's complement, a 2's complement operation is required to get the negative number into true form.

Step 3. Starting with the least significant multiplier bit, generate the partial products. When the multiplier bit is 1, the partial product is the same as the multiplicand. When the multiplier bit is 0, the partial product is zero. Shift each successive partial product one bit to the left.

Step 4. Add each successive partial product to the sum of the previous partial products to get the final product.

Step 5. If the sign bit that was determined in step 1 is negative, take the 2's complement of the product. If positive, leave the product in true form. Attach the sign bit to the product.

Example 2–22 illustrates these steps.

EXAMPLE 2–22

Multiply the signed binary numbers: 01010011 (multiplicand) and 11000101 (multiplier).

Solution **Step 1.** The sign bit of the multiplicand is 0 and the sign bit of the multiplier is 1. The sign bit of the product will be 1 (negative).

Step 2. Take the 2's complement of the multiplier to put it in true form.

$$11000101 \longrightarrow 00111011$$

Steps 3 and 4. The multiplication proceeds as follows. Notice that only the magnitude bits are used in these steps.

1010011	Multiplicand
× 0111011	Multiplier
1010011	1st partial product
+ 1010011	2nd partial product
11111001	Sum of 1st and 2nd
+ 0000000	3rd partial product
011111001	Sum
+ 1010011	4th partial product
1110010001	Sum
+ 1010011	5th partial product
100011000001	Sum
+ 1010011	6th partial product
1001100100001	Sum
+ 0000000	7th partial product
1001100100001	Final product

Step 5. Since the sign of the product is a 1 as determined in step 1, take the 2's complement of the product.

$$1001100100001 \longrightarrow 0110011011111$$

Attach the sign bit ⟶ **1 0110011011111**

Supplementary Problem Verify the multiplication is correct by converting to decimal numbers and performing the multiplication.

Division

The numbers in a division are the **dividend**, the **divisor**, and the **quotient**. These are illustrated in the following standard division format.

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

The division operation in computers is accomplished using subtraction. Since subtraction is done with an adder, division can also be accomplished with an adder.

The result of a division is called the *quotient*; the quotient is the number of times that the divisor will go into the dividend. This means that the divisor can be subtracted from the dividend a number of times equal to the quotient, as illustrated by dividing 21 by 7.

21	Dividend
− 7	1st subtraction of divisor
14	1st partial remainder
− 7	2nd subtraction of divisor
7	2nd partial remainder
− 7	3rd subtraction of divisor
0	Zero remainder

In this simple example, the divisor was subtracted from the dividend three times before a remainder of zero was obtained. Therefore, the quotient is 3.

The sign of the quotient depends on the signs of the dividend and the divisor according to the following two rules:

- **If the signs are the same, the quotient is positive.**
- **If the signs are different, the quotient is negative.**

When two binary numbers are divided, both numbers must be in true (uncomplemented) form. The basic steps in a division process are as follows:

Step 1. Determine if the signs of the dividend and divisor are the same or different. This determines what the sign of the quotient will be. Initialize the quotient to zero.

Step 2. Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient. If this partial remainder is positive, go to step 3. If the partial remainder is zero or negative, the division is complete.

Step 3. Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive, repeat for the next partial remainder. If the result is zero or negative, the division is complete.

Continue to subtract the divisor from the dividend and the partial remainders until there is a zero or a negative result. Count the number of times that the divisor is subtracted and you have the quotient. Example 2–23 illustrates these steps using 8-bit signed binary numbers.

EXAMPLE 2–23

Divide 01100100 by 00011001.

Solution **Step 1.** The signs of both numbers are positive, so the quotient will be positive. The quotient is initially zero: 00000000.

Step 2. Subtract the divisor from the dividend using 2's complement addition (remember that final carries are discarded).

01100100	Dividend
+ 11100111	2's complement of divisor
01001011	Positive 1st partial remainder

Add 1 to quotient: 00000000 + 00000001 = 00000001.

Step 3. Subtract the divisor from the 1st partial remainder using 2's complement addition.

01001011	1st partial remainder
+ 11100111	2's complement of divisor
00110010	Positive 2nd partial remainder

Add 1 to quotient: 00000001 + 00000001 = 00000010.

Step 4. Subtract the divisor from the 2nd partial remainder using 2's complement addition.

00110010	2nd partial remainder
+ 11100111	2's complement of divisor
00011001	Positive 3rd partial remainder

Add 1 to quotient: 00000010 + 00000001 = 00000011.

Step 5. Subtract the divisor from the 3rd partial remainder using 2's complement addition.

00011001	3rd partial remainder
+ 11100111	2's complement of divisor
00000000	Zero remainder

Add 1 to quotient: 00000011 + 00000001 = **00000100** (final quotient). The process is complete.

Supplementary Problem Verify that the process is correct by converting to decimal numbers and performing the division.

SECTION 2–7 REVIEW

1. List the four cases when numbers are added.
2. Add 00100001 and 10111100.
3. Subtract 00110010 from 01110111.
4. What is the sign of the product when two negative numbers are multiplied?
5. Multiply 01111111 by 00000101.
6. What is the sign of the quotient when a positive number is divided by a negative number?
7. Divide 00110000 by 00001100.

2–8 HEXADECIMAL NUMBERS

The hexadecimal number system has sixteen digits and is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal. As you are probably aware, long binary numbers are difficult to read and write because it is easy to drop or transpose a bit. Since computers and microprocessors understand only 1s and 0s, it is necessary to use these digits when you program in "machine language." Imagine writing a sixteen-bit instruction for a microprocessor system in 1s and 0s. It is much more efficient to use hexadecimal or octal; octal numbers are covered in Section 2–9. Hexadecimal is widely used in computer and microprocessor applications.

After completing this section, you should be able to

- List the hexadecimal digits
- Count in hexadecimal
- Convert from binary to hexadecimal
- Convert from hexadecimal to binary
- Convert from hexadecimal to decimal
- Convert from decimal to hexadecimal
- Add hexadecimal numbers
- Determine 2's complement of a hexadecimal number
- Subtract hexadecimal numbers

The **hexadecimal number** system has a base of sixteen; that is, it is composed of 16 digits and alphabetic **characters**. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient because each hexadecimal digit represents a 4-bit binary number (as listed in Table 2–3).

▶ **TABLE 2–3**

DECIMAL	BINARY	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Ten **numeric** digits and six alphabetic characters make up the hexadecimal number system. The use of letters A, B, C, D, E, and F to represent numbers may seem strange at first, but keep in mind that any number system is only a set of sequential symbols. If you understand what quantities these symbols represent, then the form of the symbols themselves is less important once you get accustomed to using them. We will use the subscript 16 to designate hexadecimal numbers to avoid confusion with decimal numbers. Sometimes, you may see an "h" following a hexadecimal number.

Counting in Hexadecimal

How do you count in hexadecimal once you get to F? Simply start over with another column and continue as follows:

> 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, . . .

With two hexadecimal digits, you can count up to FF_{16}, which is decimal 255. To count beyond this, three hexadecimal digits are needed. For instance, 100_{16} is decimal 256, 101_{16} is decimal 257, and so forth. The maximum 3-digit hexadecimal number is FFF_{16}, or decimal 4095. The maximum 4-digit hexadecimal number is $FFFF_{16}$, which is decimal 65,535.

Binary-to-Hexadecimal Conversion

Converting a binary number to hexadecimal is a very straightforward procedure. Simply break the binary number into 4-bit groups, starting at the right-most bit and replace each 4-bit group with the equivalent hexadecimal symbol as illustrated in Example 2–24.

EXAMPLE 2–24

Convert the following binary numbers to hexadecimal:

(a) 1100101001010111 **(b)** 111111000101101001

Solution **(a)** $\underbrace{1100}_{\downarrow\,C}\underbrace{1010}_{\downarrow\,A}\underbrace{0101}_{\downarrow\,5}\underbrace{0111}_{\downarrow\,7} = \mathbf{CA57_{16}}$ **(b)** $\underbrace{0011}_{\downarrow\,3}\underbrace{1111}_{\downarrow\,F}\underbrace{0001}_{\downarrow\,1}\underbrace{0110}_{\downarrow\,6}\underbrace{1001}_{\downarrow\,9} = \mathbf{3F169_{16}}$

Two zeros have been added in part (b) to complete a 4-bit group at the left.

Supplementary Problem Convert the binary number 10011111011110011100 to hexadecimal.

Hexadecimal-to-Binary Conversion

To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits as illustrated in Example 2–25.

EXAMPLE 2–25

Determine the binary numbers for the following hexadecimal numbers:

(a) $10A4_{16}$ **(b)** $CF8E_{16}$ **(c)** 9742_{16}

Solution **(a)** $\overset{1\,\downarrow}{1}\overbrace{0000}^{0\,\downarrow}\overbrace{1010}^{A\,\downarrow}\overbrace{0100}^{4\,\downarrow}$ **(b)** $\overbrace{1100}^{C\,\downarrow}\overbrace{1111}^{F\,\downarrow}\overbrace{1000}^{8\,\downarrow}\overbrace{1110}^{E\,\downarrow}$ **(c)** $\overbrace{1001}^{9\,\downarrow}\overbrace{0111}^{7\,\downarrow}\overbrace{0100}^{4\,\downarrow}\overbrace{0010}^{2\,\downarrow}$

In part (a), the MSB is understood to have three zeros preceding it, thus forming a 4-bit group.

Supplementary Problem Convert the hexadecimal number 6BD3 to binary.

It should be clear that it is much easier to deal with a hexadecimal number than with the equivalent binary number. Since conversion is so easy, the hexadecimal system is widely used for representing binary numbers in programming, printouts, and displays.

Hexadecimal-to-Decimal Conversion

One way to find the decimal equivalent of a hexadecimal number is to first convert the hexadecimal number to binary and then convert from binary to decimal. Example 2–26 illustrates this procedure.

EXAMPLE 2–26

Convert the following hexadecimal numbers to decimal:

(a) $1C_{16}$ **(b)** $A85_{16}$

Solution Remember, convert the hexadecimal number to binary first, then to decimal.

(a)

$$\begin{array}{cc} 1 & C \\ \downarrow & \downarrow \\ \overbrace{0001} & \overbrace{1100} \end{array} = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = \mathbf{28_{10}}$$

(b)

$$\begin{array}{ccc} A & 8 & 5 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{1010} & \overbrace{1000} & \overbrace{0101} \end{array} = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = \mathbf{2693_{10}}$$

Supplementary Problem Convert the hexadecimal number 6BD to decimal.

Another way to convert a hexadecimal number to its decimal equivalent is to multiply the decimal value of each hexadecimal digit by its weight and then take the sum of these products. The weights of a hexadecimal number are increasing powers of 16 (from right to left). For a 4-digit hexadecimal number, the weights are

$$\begin{array}{cccc} 16^3 & 16^2 & 16^1 & 16^0 \\ 4096 & 256 & 16 & 1 \end{array}$$

Example 2–27 shows this conversion method.

EXAMPLE 2–27

Convert the following hexadecimal numbers to decimal:

(a) $E5_{16}$ **(b)** $B2F8_{16}$

Solution Recall from Table 2–3 that letters A through F represent decimal numbers 10 through 15, respectively.

(a) $E5_{16} = (E \times 16) + (5 \times 1) = (14 \times 16) + (5 \times 1) = 224 + 5 = \mathbf{229_{10}}$

(b)

$$\begin{aligned} B2F8_{16} &= (B \times 4096) + (2 \times 256) + (F \times 16) + (8 \times 1) \\ &= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1) \\ &= 45{,}056 + 512 + 240 + 8 = \mathbf{45{,}816_{10}} \end{aligned}$$

Supplementary Problem Convert $60A_{16}$ to decimal.

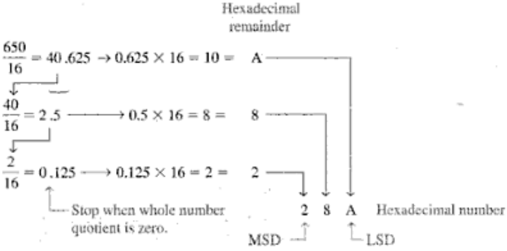
Decimal-to-Hexadecimal Conversion

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions. The first remainder produced is the least significant digit (LSD). Each successive division by 16 yields a remainder that becomes a digit in the equivalent hexadecimal number. This procedure is similar to repeated division by 2 for decimal-to-binary conversion that was covered in Section 2–3. Example 2–28 illustrates the procedure. Note that when a quotient has a fractional part, the fractional part is multiplied by the divisor to get the remainder.

EXAMPLE 2–28

Convert the decimal number 650 to hexadecimal by repeated division by 16.

Solution


Hexadecimal remainder
$\frac{650}{16} = 40.625 \rightarrow 0.625 \times 16 = 10 =$ A
$\frac{40}{16} = 2.5 \longrightarrow 0.5 \times 16 = 8 =$ 8
$\frac{2}{16} = 0.125 \longrightarrow 0.125 \times 16 = 2 =$ 2
Stop when whole number quotient is zero.
2 8 A Hexadecimal number
MSD
LSD

Supplementary Problem Convert decimal 2591 to hexadecimal.

Hexadecimal Addition

Hexadecimal is a convenient way to represent binary numbers.

Addition can be done directly with hexadecimal numbers by remembering that the hexadecimal digits 0 through 9 are equivalent to decimal digits 0 through 9 and that hexadecimal digits A through F are equivalent to decimal numbers 10 through 15. When adding two hexadecimal numbers, use the following rules. (Decimal numbers are indicated by a subscript 10.)

1. In any given column of an addition problem, think of the two hexadecimal digits in terms of their decimal values. For instance, $5_{16} = 5_{10}$ and $C_{16} = 12_{10}$.
2. If the sum of these two digits is 15_{10} or less, bring down the corresponding hexadecimal digit.
3. If the sum of these two digits is greater than 15_{10}, bring down the amount of the sum that exceeds 16_{10} and carry a 1 to the next column.

EXAMPLE 2–29

Add the following hexadecimal numbers:

(a) $23_{16} + 16_{16}$ **(b)** $58_{16} + 22_{16}$ **(c)** $2B_{16} + 84_{16}$ **(d)** $DF_{16} + AC_{16}$

Solution **(a)**

$$\begin{array}{r} 23_{16} \\ +16_{16} \\ \hline \mathbf{39_{16}} \end{array}$$

right column: $3_{16} + 6_{16} = 3_{10} + 6_{10} = 9_{10} = 9_{16}$

left column: $2_{16} + 1_{16} = 2_{10} + 1_{10} = 3_{10} = 3_{16}$

(b)

$$\begin{array}{r} 58_{16} \\ +22_{16} \\ \hline \mathbf{7A_{16}} \end{array}$$

right column: $8_{16} + 2_{16} = 8_{10} + 2_{10} = 10_{10} = A_{16}$

left column: $5_{16} + 2_{16} = 5_{10} + 2_{10} = 7_{10} = 7_{16}$

(c)

$$\begin{array}{r} 2B_{16} \\ +84_{16} \\ \hline \mathbf{AF_{16}} \end{array}$$

right column: $B_{16} + 4_{16} = 11_{10} + 4_{10} = 15_{10} = F_{16}$

left column: $2_{16} + 8_{16} = 2_{10} + 8_{10} = 10_{10} = A_{16}$

(d)

$$\begin{array}{r} DF_{16} \\ +AC_{16} \\ \hline \mathbf{18B_{16}} \end{array}$$

right column: $F_{16} + C_{16} = 15_{10} + 12_{10} = 27_{10}$
$27_{10} - 16_{10} = 11_{10} = B_{16}$ with a 1 carry

left column: $D_{16} + A_{16} + 1_{16} = 13_{10} + 10_{10} + 1_{10} = 24_{10}$
$24_{10} - 16_{10} = 8_{10} = 8_{16}$ with a 1 carry

Supplementary Problem Add $4C_{16}$ and $3A_{16}$.

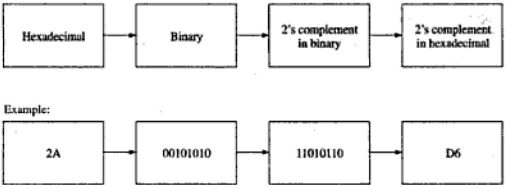
Hexadecimal Subtraction

As you have learned, the 2's complement allows you to subtract by adding binary numbers. Since a hexadecimal number can be used to represent a binary number, it can also be used to represent the 2's complement of a binary number.

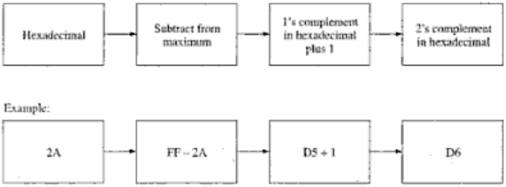
There are three ways to get the 2's complement of a hexadecimal number. Method 1 is the most common and easiest to use. Methods 2 and 3 are alternate methods.

Method 1. Convert the hexadecimal number to binary. Take the 2's complement of the binary number. Convert the result to hexadecimal. This is illustrated in Figure 2–3.

Method 2. Subtract the hexadecimal number from the maximum hexadecimal number and add 1. This is illustrated in Figure 2–4.



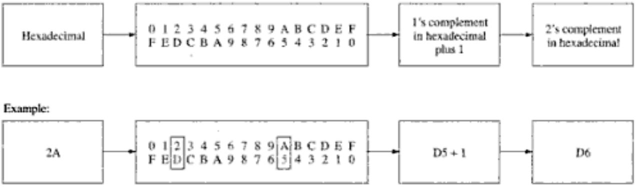
▲ FIGURE 2–3



Hexadecimal
Subtract from maximum
1's complement in hexadecimal plus 1
2's complement in hexadecimal
Example:
2A
FF − 2A
D5 + 1
D6

▲ **FIGURE 2–4**

Method 3. Write the sequence of single hexadecimal digits. Write the sequence in reverse below the forward sequence. The 1's complement of each hex digit is the digit directly below it. Add 1 to the resulting number to get the 2's complement. This is illustrated in Figure 2–5.



Hexadecimal
0 1 2 3 4 5 6 7 8 9 A B C D E F
F E D C B A 9 8 7 6 5 4 3 2 1 0
1's complement in hexadecimal plus 1
2's complement in hexadecimal
Example:
2A
0 1 2 3 4 5 6 7 8 9 A B C D E F
F E D C B A 9 8 7 6 5 4 3 2 1 0
D5 + 1
D6

▲ **FIGURE 2–5**

EXAMPLE 2–30

Subtract the following hexadecimal numbers:

(a) $84_{16} - 2A_{16}$ **(b)** $C3_{16} - 0B_{16}$

Solution **(a)** $2A_{16} = 00101010$

2's complement of $2A_{16} = 11010110 = D6_{16}$ (using Method 1)

$$\begin{array}{rl} 84_{16} & \\ \underline{+\ D6_{16}} & \text{Add} \\ \cancel{1}5A_{16} & \text{Drop carry, as in 2's complement addition} \end{array}$$

The difference is **$5A_{16}$**.

(b) $0B_{16} = 00001011$

2's complement of $0B_{16} = 11110101 = F5_{16}$ (using Method 1)

$$\begin{array}{rl} C3_{16} & \\ \underline{+\ F5_{16}} & \text{Add} \\ \cancel{1}B8_{16} & \text{Drop carry} \end{array}$$

The difference is **B8**$_{16}$.

Supplementary Problem Subtract 173_{16} from BCD_{16}.

SECTION 2–8 REVIEW

1. Convert the following binary numbers to hexadecimal:
   (a) 10110011 (b) 110011101000
2. Convert the following hexadecimal numbers to binary:
   (a) 57_{16} (b) $3A5_{16}$ (c) $F80B_{16}$
3. Convert $9B30_{16}$ to decimal.
4. Convert the decimal number 573 to hexadecimal.
5. Add the following hexadecimal numbers directly:
   (a) $18_{16} + 34_{16}$ (b) $3F_{16} + 2A_{16}$
6. Subtract the following hexadecimal numbers:
   (a) $75_{16} - 21_{16}$ (b) $94_{16} - 5C_{16}$

2–9 OCTAL NUMBERS

Like the hexadecimal number system, the octal number system provides a convenient way to express binary numbers and codes. However, it is used less frequently than hexadecimal in conjunction with computers and microprocessors to express binary quantities for input and output purposes.

After completing this section, you should be able to

- Write the digits of the octal number system
- Convert from octal to decimal
- Convert from decimal to octal
- Convert from octal to binary
- Convert from binary to octal

The octal number system is composed of eight digits, which are

0, 1, 2, 3, 4, 5, 6, 7. It has a base of 8.

To count above 7, begin another column and start over:

10, 11, 12, 13, 14, 15, 16, 17, 20, 21, . . .

Counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used. To distinguish octal numbers from decimal numbers or hexadecimal numbers, we will use the subscript 8 to indicate an octal number. For instance, 15_8 in octal is equivalent to 13_{10} in decimal and D in hexadecimal. Sometimes, you may see an "o"or a "Q" following an octal number.

Octal-to-Decimal Conversion

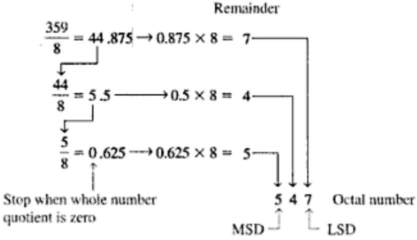
Since the octal number system has a base of eight, each successive digit position is an increasing power of eight, beginning in the right-most column with 8^0. The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, as illustrated here for 2374_8.

$$\begin{array}{rcccc} \text{Weight:} & 8^3 & 8^2 & 8^1 & 8^0 \\ \text{Octal number:} & 2 & 3 & 7 & 4 \end{array}$$

$$\begin{aligned} 2374_8 &= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0) \\ &= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1) \\ &= 1024 + 192 + 56 + 4 = 1276_{10} \end{aligned}$$

Decimal-to-Octal Conversion

A method of converting a decimal number to an octal number is the repeated division-by-8 method, which is similar to the method used in the conversion of decimal numbers to binary or to hexadecimal. To show how it works, let us convert the decimal number 359 to octal. Each successive division by 8 yields a remainder that becomes a digit in the equivalent octal number. The first remainder generated is the least significant digit (LSD).



Octal-to-Binary Conversion

Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary. Each octal digit is represented by three bits as shown in Table 2–4.

TABLE 2-4

Octal/binary conversion

OCTAL DIGIT	0	1	2	3	4	5	6	7
BINARY	000	001	010	011	100	101	110	111

To convert an octal number to a binary number, simply replace each octal digit with the appropriate three bits. This procedure is illustrated in Example 2–31.

EXAMPLE 2–31

Convert each of the following octal numbers to binary:

(a) 13_8 **(b)** 25_8 **(c)** 140_8 **(d)** 7526_8

Solution

(a) $\begin{array}{cc} 1 & 3 \\ \downarrow & \downarrow \\ \overbrace{001} & \overbrace{011} \end{array}$ **(b)** $\begin{array}{cc} 2 & 5 \\ \downarrow & \downarrow \\ \overbrace{010} & \overbrace{101} \end{array}$ **(c)** $\begin{array}{ccc} 1 & 4 & 0 \\ \downarrow & \downarrow & \downarrow \\ \overbrace{001} & \overbrace{100} & \overbrace{000} \end{array}$ **(d)** $\begin{array}{cccc} 7 & 5 & 2 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \overbrace{111} & \overbrace{101} & \overbrace{010} & \overbrace{110} \end{array}$

Supplementary Problem Convert each of the binary numbers to decimal and verify that each value agrees with the decimal value of the corresponding octal number.

Binary-to-Octal Conversion

Conversion of a binary number to an octal number is the reverse of the octal-to-binary conversion. The procedure is as follows: Start with the right-most group of three bits and, moving from right to left, convert each 3-bit group to the equivalent octal digit. If there are not three bits available for the left-most group, add either one or two zeros to make a complete group. These leading zeros do not affect the value of the binary number.

EXAMPLE 2–32

Convert each of the following binary numbers to octal:

(a) 110101 **(b)** 101111001 **(c)** 100110011010 **(d)** 11010000100

Solution

(a) $\begin{array}{cc} \underbrace{110} & \underbrace{101} \\ \downarrow & \downarrow \\ 6 & 5 \end{array} = \mathbf{65_8}$ **(b)** $\begin{array}{ccc} \underbrace{101} & \underbrace{111} & \underbrace{001} \\ \downarrow & \downarrow & \downarrow \\ 5 & 7 & 1 \end{array} = \mathbf{571_8}$

(c) $\begin{array}{cccc} \underbrace{100} & \underbrace{110} & \underbrace{011} & \underbrace{010} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 4 & 6 & 3 & 2 \end{array} = \mathbf{4632_8}$ **(d)** $\begin{array}{cccc} \underbrace{011} & \underbrace{010} & \underbrace{000} & \underbrace{100} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 3 & 2 & 0 & 4 \end{array} = \mathbf{3204_8}$

Supplementary Problem Convert the binary number 1010101000111110010 to octal.

SECTION 2–9 REVIEW

1. Convert the following octal numbers to decimal:
   (a) 73_8 (b) 125_8
2. Convert the following decimal numbers to octal:
   (a) 98_{10} (b) 163_{10}
3. Convert the following octal numbers to binary:
   (a) 46_8 (b) 723_8 (c) 5624_8
4. Convert the following binary numbers to octal:
   (a) 110101111 (b) 1001100010 (c) 10111111001

2–10 BINARY CODED DECIMAL (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in the BCD system, so it is very easy to convert between decimal and BCD. Because we like to read and write in decimal, the BCD code provides an excellent interface to binary systems. Examples of such interfaces are keypad inputs and digital readouts.

After completing this section, you should be able to

- Convert each decimal digit to BCD
- Express decimal numbers in BCD
- Convert from BCD to decimal
- Add BCD numbers

The 8421 Code

The 8421 code is a type of **BCD** (binary coded decimal) code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of four bits. The designation 8421 indicates the binary weights of the four bits (2^3, 2^2, 2^1, 2^0). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in Table 2–5. The 8421 code is the predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

TABLE 2–5

Decimal/BCD conversion

DECIMAL DIGIT	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Invalid Codes You should realize that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used—1010, 1011, 1100, 1101, 1110, and 1111—are invalid in the 8421 BCD code.

In BCD, 4 bits represent each decimal digit.

To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code, as shown by Example 2–33.

EXAMPLE 2–33

Convert each of the following decimal numbers to BCD:

(a) 35 **(b)** 98 **(c)** 170 **(d)** 2469

Solution

(a)
$$\underset{\textbf{00110101}}{\underline{\begin{array}{cc} 3 & 5 \\ \downarrow & \downarrow \end{array}}}$$

(b)
$$\underset{\textbf{10011000}}{\underline{\begin{array}{cc} 9 & 8 \\ \downarrow & \downarrow \end{array}}}$$

(c)
$$\underset{\textbf{000101110000}}{\underline{\begin{array}{ccc} 1 & 7 & 0 \\ \downarrow & \downarrow & \downarrow \end{array}}}$$

(d)
$$\underset{\textbf{0010010001101001}}{\underline{\begin{array}{cccc} 2 & 4 & 6 & 9 \\ \downarrow & \downarrow & \downarrow & \downarrow \end{array}}}$$

Supplementary Problem **Convert the decimal number 9673 to BCD.**

It is equally easy to determine a decimal number from a BCD number. Start at the right-most bit and break the code into groups of four bits. Then, write the decimal digit represented by each 4-bit group. Example 2–34 illustrates.

EXAMPLE 2–34

Convert each of the following BCD codes to decimal:

(a) 10000110 **(b)** 001101010001 **(c)** 1001010001110000

Solution

(a) $\underbrace{1000}_{\substack{\downarrow\\ \mathbf{8}}}\underbrace{0110}_{\substack{\downarrow\\ \mathbf{6}}}$ **(b)** $\underbrace{0011}_{\substack{\downarrow\\ \mathbf{3}}}\underbrace{0101}_{\substack{\downarrow\\ \mathbf{5}}}\underbrace{0001}_{\substack{\downarrow\\ \mathbf{1}}}$ **(c)** $\underbrace{1001}_{\substack{\downarrow\\ \mathbf{9}}}\underbrace{0100}_{\substack{\downarrow\\ \mathbf{4}}}\underbrace{0111}_{\substack{\downarrow\\ \mathbf{7}}}\underbrace{0000}_{\substack{\downarrow\\ \mathbf{0}}}$

Supplementary Problem Convert the BCD code 10000010001001110110 to decimal.

BCD Addition

BCD is a numerical code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step 1. Add the two BCD numbers, using the rules for binary addition in Section 2–4.

Step 2. If a 4-bit sum is equal to or less than 9, it is a valid BCD number.

Step 3. If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Example 2–35 illustrates BCD additions in which the sum in each 4-bit column is equal to or less than 9, and the 4-bit sums are therefore valid BCD numbers. Example 2–36 illustrates the procedure in the case of invalid sums (greater than 9 or a carry).

EXAMPLE 2–35

Add the following BCD numbers:

(a) 0011 + 0100 **(b)** 00100011 + 00010101

(c) 10000110 + 00010011 **(d)** 010001010000 + 010000010111

Solution The decimal number additions are shown for comparison.

(a) $\begin{array}{r} 0011\\ +\,0100\\ \hline \mathbf{0111}\end{array}\qquad\begin{array}{r} 3\\ +\,4\\ \hline 7\end{array}$ **(b)** $\begin{array}{r} 0010\ \ 0011\\ +\,0001\ \ 0101\\ \hline \mathbf{0011\ \ 1000}\end{array}\qquad\begin{array}{r} 23\\ +\,15\\ \hline 38\end{array}$

(c) $\begin{array}{r} 1000\ \ 0110\\ +\,0001\ \ 0011\\ \hline \mathbf{1001\ \ 1001}\end{array}\qquad\begin{array}{r} 86\\ +\,13\\ \hline 99\end{array}$ **(d)** $\begin{array}{r} 0100\ \ 0101\ \ 0000\\ +\,0100\ \ 0001\ \ 0111\\ \hline \mathbf{1000\ \ 0110\ \ 0111}\end{array}\qquad\begin{array}{r} 450\\ +\,417\\ \hline 867\end{array}$

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Supplementary Problem Add the BCD numbers: 1001000001000011 + 0000100100100101.

EXAMPLE 2–36

Add the following BCD numbers:

(a) 1001 + 0100
(b) 1001 + 1001
(c) 00010110 + 00010101
(d) 01100111 + 01010011

Solution The decimal number additions are shown for comparison.

(a)

	1001		9
	+ 0100		+4
	1101	Invalid BCD number (>9)	13
	+ 0110	Add 6	
0001	**0011**	Valid BCD number	
↓	↓		
1	3		

(b)

	1001		9
	+ 1001		+ 9
1	0010	Invalid because of carry	18
	+ 0110	Add 6	
0001	**1000**	Valid BCD number	
↓	↓		
1	8		

(c)

0001	0110		16
+ 0001	0101		+ 15
0010	1011	Right group is invalid (>9), left group is valid.	31
	+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
0011	**0001**	Valid BCD number	
↓	↓		
3	1		

(d)

	0110	0111		67
	+ 0101	0011		+ 53
	1011	1010	Both groups are invalid (>9)	120
	+ 0110	+ 0110	Add 6 to both groups	
0001	**0010**	**0000**	Valid BCD number	
↓	↓	↓		
1	2	0		

Supplementary Problem Add the BCD numbers: 01001000 + 00110100.

SECTION 2–10 REVIEW

1. What is the binary weight of each 1 in the following BCD numbers?
   (a) 0010 (b) 1000 (c) 0001 (d) 0100
2. Convert the following decimal numbers to BCD:
   (a) 6 (b) 15 (c) 273 (d) 849
3. What decimal numbers are represented by each BCD code?
   (a) 10001001 (b) 001001111000 (c) 000101010111
4. In BCD addition, when is a 4-bit sum invalid?

2–11 DIGITAL CODES

There are many specialized codes used in digital systems. You have just learned about the BCD code; now let us look at a few others. Some codes are strictly numeric, like BCD, and others are alphanumeric; that is, they are used to represent numbers, letters, symbols, and instructions. The codes introduced in this section are the Gray code and the ASCII code. Also, the detection of errors in codes using a parity bit is covered. Information on codes that use other methods of error detection and an error correcting code are in Appendix B.

After completing this section, you should be able to

- Explain the advantage of the Gray code
- Convert between Gray code and binary
- Use the ASCII code
- Identify errors in codes based on the parity method

The Gray Code

The single bit change characterisitic of the Gray code minimizes the chance for error.

The **Gray code** is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that *it exhibits only a single bit change from one code word to the next in sequence.* This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence.

Table 2–6 is a listing of the 4-bit Gray code for decimal numbers 0 through 15. Binary numbers are shown in the table for reference. Like binary numbers, *the Gray code can have any number of bits.* Notice the single-bit change between successive Gray code words. For instance, in going from decimal 3 to decimal 4, the Gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change is in the third bit from the right in the Gray code; the others remain the same.

▼ TABLE 2–6

Four-bit Gray code

DECIMAL	BINARY	GRAY CODE	DECIMAL	BINARY	GRAY CODE
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Binary-to-Gray Code Conversion Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

For example, the conversion of the binary number 10110 to Gray code is as follows:

$$\begin{array}{cccccccccl} 1 & -+\rightarrow & 0 & -+\rightarrow & 1 & -+\rightarrow & 1 & -+\rightarrow & 0 & \text{Binary} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 1 & & 1 & & 1 & & 0 & & 1 & \text{Gray} \end{array}$$

The Gray code is 11101.

***Gray-to-Binary** Conversion* To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

For example, the conversion of the Gray code word 11011 to binary is as follows:

$$\begin{array}{cccccccccl} 1 & & 1 & & 0 & & 1 & & 1 & \text{Gray} \\ \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \\ 1 & & 0 & & 0 & & 1 & & 0 & \text{Binary} \end{array}$$

The binary number is 10010.

EXAMPLE 2–37

(a) Convert the binary number 11000110 to Gray code.

(b) Convert the Gray code 10101111 to binary.

Solution (a) Binary to Gray code:

$$\begin{array}{ccccccccccccccc} 1 & -+\rightarrow & 1 & -+\rightarrow & 0 & -+\rightarrow & 0 & -+\rightarrow & 0 & -+\rightarrow & 1 & -+\rightarrow & 1 & -+\rightarrow & 0 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \mathbf{1} & & \mathbf{0} & & \mathbf{1} & & \mathbf{0} & & \mathbf{0} & & \mathbf{1} & & \mathbf{0} & & \mathbf{1} \end{array}$$

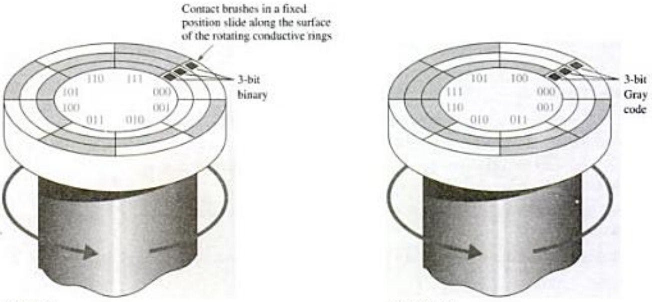
(b) Gray code to binary:

$$\begin{array}{ccccccccccccccc} 1 & & 0 & & 1 & & 0 & & 1 & & 1 & & 1 & & 1 \\ \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow & \overset{+}{\nearrow} & \downarrow \\ \mathbf{1} & & \mathbf{1} & & \mathbf{0} & & \mathbf{0} & & \mathbf{1} & & \mathbf{0} & & \mathbf{1} & & \mathbf{0} \end{array}$$

Supplementary Problem (a) Convert binary 101101 to Gray code. (b) Convert Gray code 100111 to binary.

Application Example

A simplified diagram of a 3-bit shaft position encoder mechanism is shown in Figure 2–6. Basically, there are three concentric conductive rings that are segmented into eight sectors. The more sectors there are, the more accurately the position can be represented, but we are using only eight for purposes of illustration. Each sector of each ring is fixed at either a high-level or a low-level voltage to represent 1s and 0s. A 1 is indicated by a color sector and a 0 by a white sector. As the rings rotate with the shaft, they make contact with a brush arrangement, that is, in a fixed position and to which output lines are connected. As the shaft rotates counterclockwise through 360°, the eight sectors move past the three brushes producing a 3-bit binary output that indicates the shaft position.


Contact brushes in a fixed position slide along the surface of the rotating conductive rings
3-bit binary
110
111
101
000
100
001
011
010
3-bit Gray code
101
100
111
000
110
001
010
011

(a) Binary

(b) Gray code

▲ **FIGURE 2–6**

In Figure 2–6(a), the sectors are arranged in a straight binary pattern, so that the brushes go from 000 to 001 to 010 to 011, and so on. When the brushes are on color sectors, they output a 1 and when on white sectors, they output a 0. If one brush is slightly ahead of the others during the transition from one sector to the next, an erroneous output can occur. Consider what happens when the brushes are on the 111 sector and about to enter the 000 sector. If the MSB brush is slightly ahead, the position would be incorrectly indicated by a transitional 011 instead of a 111 or a 000. In this type of application, it is virtually impossible to maintain precise mechanical alignment of all the brushes; therefore, some error will always occur at many of the transitions between sectors.

The Gray code is used to eliminate the error problem which is inherent in the binary code. As shown in Figure 2–6(b), the Gray code assures that only one bit will change between adjacent sectors. This means that even though the brushes may not be in precise alignment, there will never be a transitional error. For example, let's again consider what happens when the brushes are on the 111 sector and about to move into the next sector, 101. The only two possible outputs during the transition are 111 and 101, no matter how the brushes are aligned. A similar situation occurs at the transitions between each of the other sectors.

Alphanumeric Codes

In order to communicate, you need not only numbers, but also letters and other symbols. In the strictest sense, **alphanumeric** codes are codes that represent numbers and alphabetic characters (letters). Most such codes, however, also represent other characters such as symbols and various instructions necessary for conveying information.

At a minimum, an alphanumeric code must represent 10 decimal digits and 26 letters of the alphabet, for a total of 36 items. This number requires six bits in each code combination because five bits are insufficient ($2^5 = 32$). There are 64 total combinations of six bits, so there are 28 unused code combinations. Obviously, in many applications, symbols other than just numbers and letters are necessary to communicate completely. You need spaces, periods, colons, semicolons, question marks, etc. You also need instructions to tell the receiving system what to do with the information. So, with codes that are six bits long, you can handle decimal numbers, the alphabet, and 28 other symbols. This should give you an idea of the requirements for a basic alphanumeric code. The ASCII is the most common alphanumeric code and is covered next.

ASCII

ASCII is the abbreviation for American Standard Code for Information Interchange. Pronounced "askee," ASCII is a universally accepted alphanumeric code used in most computers and other electronic equipment. Most computer keyboards are standardized with the ASCII. When you enter a letter, a number, or control command, the corresponding ASCII code goes into the computer.

ASCII has 128 characters and symbols represented by a 7-bit binary code. Actually, ASCII can be considered an 8-bit code with the MSB always 0. This 8-bit code is 00 through 7F in hexadecimal. The first thirty-two ASCII characters are nongraphic commands that are never printed or displayed and are used only for control purposes. Examples of the control characters are "null," "line feed," "start of text," and "escape." The other characters are graphic symbols that can be printed or displayed and include the letters of the alphabet (lowercase and uppercase), the ten decimal digits, punctuation signs, and other commonly used symbols.

Table 2–7 is a listing of the ASCII code showing the decimal, hexadecimal, and binary representations for each character and symbol. The left section of the table lists the names of the 32 control characters (00 through 1F hexadecimal). The graphic symbols are listed in the rest of the table (20 through 7F hexadecimal).

The ASCII Control Characters The first thirty-two codes in the ASCII table (Table 2–7) represent the control characters. These are used to allow devices such as a computer and printer

EXAMPLE 2–38

Determine the binary ASCII codes that are entered from the computer's keyboard when the following BASIC program statement is typed in. Also, express each code in hexadecimal.

```
20 PRINT "A="; X
```

Solution The ASCII code for each symbol is found in Table 2–7.

Symbol	Binary	Hexadecimal
2	0110010	32_{16}
0	0110000	30_{16}
Space	0100000	20_{16}
P	1010000	50_{16}
R	1010010	52_{16}
I	1001001	49_{16}
N	1001110	$4E_{16}$
T	1010100	54_{16}
Space	0100000	20_{16}
"	0100010	22_{16}
A	1000001	41_{16}
=	0111101	$3D_{16}$
"	0100010	22_{16}
;	0111011	$3B_{16}$
X	1011000	58_{16}

Supplementary Problem Determine the sequence of ASCII codes required for the following program statement and express them in hexadecimal:

```
80 INPUT Y
```

▼ TABLE 2–7

CONTROL CHARACTERS				GRAPHIC SYMBOLS											
NAME	DEC	BINARY	HEX	SYMBOL	DEC	BINARY	HEX	SYMBOL	DEC	BINARY	HEX	SYMBOL	DEC	BINARY	HEX
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	`	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	-	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

to communicate with each other when passing information and data. Table 2–8 lists the control characters and the control key function that allows them to be entered directly from an ASCII keyboard by pressing the control key (CTRL) and the corresponding symbol. A brief description of each control character is also given.

Extended ASCII Characters

In addition to the 128 standard ASCII characters, there are an additional 128 characters that were adopted by IBM for use in their PCs (personal computers). Because of the popularity of the PC, these particular extended ASCII characters are also used in applications other than PCs and have become essentially an unofficial standard.

The extended ASCII characters are represented by an 8-bit code series from hexadecimal 80 to hexadecimal FF.

TABLE 2–8

NAME	DECIMAL	HEX	KEY	DESCRIPTION
NUL	0	00	CTRL @	null character
SOH	1	01	CTRL A	start of header
STX	2	02	CTRL B	start of text
ETX	3	03	CTRL C	end of text
EOT	4	04	CTRL D	end of transmission
ENQ	5	05	CTRL E	enquire
ACK	6	06	CTRL F	acknowledge
BEL	7	07	CTRL G	bell
BS	8	08	CTRL H	backspace
HT	9	09	CTRL I	horizontal tab
LF	10	0A	CTRL J	line feed
VT	11	0B	CTRL K	vertical tab
FF	12	0C	CTRL L	form feed (new page)
CR	13	0D	CTRL M	carriage return
SO	14	0E	CTRL N	shift out
SI	15	0F	CTRL O	shift in
DLE	16	10	CTRL P	data link escape
DC1	17	11	CTRL Q	device control 1
DC2	18	12	CTRL R	device control 2
DC3	19	13	CTRL S	device control 3
DC4	20	14	CTRL T	device control 4
NAK	21	15	CTRL U	negative acknowledge
SYN	22	16	CTRL V	synchronize
ETB	23	17	CTRL W	end of transmission block
CAN	24	18	CTRL X	cancel
EM	25	19	CTRL Y	end of medium
SUB	26	1A	CTRL Z	substitute
ESC	27	1B	CTRL [	escape
FS	28	1C	CTRL /	file separator
GS	29	1D	CTRL]	group separator
RS	30	1E	CTRL ^	record separator
US	31	1F	CTRL _	unit separator

The extended ASCII contains characters in the following general categories:

1. Foreign (non-English) alphabetic characters
2. Foreign currency symbols
3. Greek letters
4. Mathematical symbols
5. Drawing characters
6. Bar graphing characters
7. Shading characters

Table 2–9 is a list of the extended ASCII character set with the decimal and hexadecimal representations.

▼ **TABLE 2-9**

SYMBOL	DEC	HEX	SYMBOL	DEC	HEX	SYMBOL	DEC	HEX	SYMBOL	DEC	HEX
Ç	128	80	á	160	A0	└	192	C0	α	224	E0
ü	129	81	í	161	A1	┴	193	C1	β	225	E1
é	130	82	ó	162	A2	┬	194	C2	Γ	226	E2
â	131	83	ú	163	A3	├	195	C3	π	227	E3
ä	132	84	ñ	164	A4	─	196	C4	Σ	228	E4
à	133	85	Ñ	165	A5	┼	197	C5	σ	229	E5
å	134	86	ª	166	A6	╞	198	C6	µ	230	E6
ç	135	87	º	167	A7	╟	199	C7	τ	231	E7
ê	136	88	¿	168	A8	╚	200	C8	Φ	232	E8
ë	137	89	⌐	169	A9	╔	201	C9	Θ	233	E9
è	138	8A	¬	170	AA	╩	202	CA	Ω	234	EA
ï	139	8B	½	171	AB	╦	203	CB	δ	235	EB
î	140	8C	¼	172	AC	╠	204	CC	∞	236	EC
ì	141	8D	¡	173	AD	═	205	CD	φ	237	ED
Ä	142	8E	«	174	AE	╬	206	CE	ε	238	EE
Å	143	8F	»	175	AF	╧	207	CF	∩	239	EF
É	144	90	░	176	B0	╨	208	D0	≡	240	F0
æ	145	91	▒	177	B1	╤	209	D1	±	241	F1
Æ	146	92	▓	178	B2	╥	210	D2	≥	242	F2
ô	147	93	│	179	B3	╙	211	D3	≤	243	F3
ö	148	94	┤	180	B4	╘	212	D4	⌠	244	F4
ò	149	95	╡	181	B5	╒	213	D5	⌡	245	F5
û	150	96	╢	182	B6	╓	214	D6	÷	246	F6
ù	151	97	╖	183	B7	╫	215	D7	≈	247	F7
ÿ	152	98	╕	184	B8	╪	216	D8	°	248	F8
Ö	153	99	╣	185	B9	┘	217	D9	•	249	F9
Ü	154	9A	║	186	BA	┌	218	DA	·	250	FA
¢	155	9B	╗	187	BB	█	219	DB	√	251	FB
£	156	9C	╝	188	BC	▄	220	DC	η	252	FC
¥	157	9D	╜	189	BD	▌	221	DD	²	253	FD
Pt	158	9E	╛	190	BE	▐	222	DE	■	254	FE
ƒ	159	9F	┐	191	BF	▀	223	DF		255	FF

SECTION 2–11 REVIEW

1. Convert the following binary numbers to the Gray code:
   (a) 1100 (b) 1010 (c) 11010
2. Convert the following Gray codes to binary:
   (a) 1000 (b) 1010 (c) 11101
3. What is the ASCII representation for each of the following characters? Express each as a bit pattern and in hexadecimal notation.
   (a) K (b) r (c) $ (d) +

2–12 ERROR DETECTION AND CORRECTION CODES

When digital signals (groups of 0s and 1s) are transmitted from one circuit or system to another circuit or system, error may occur during transmission. The change of a 1 to 0 or a 0 to 1 during transmission is known as error. It is necessary to detect and correct the error to obtain a correct message.

A number of codes exist for detection and correction of error in digital transmission. The concept of parity of a group of bits or digital word is the key for error detection and correction.

After completing this section you should be able to

- Understand concept of parity
- Apply parity method of erro detection
- Make use of various error detection codes
- Use Hamming code

Parity Method for Error Detection

Many systems use a parity bit as a means for bit **error detection**. Any group of bits contain either an even or an odd number of 1s. A **parity** bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

A given system operates with even or odd parity, but not both. For instance, if a system operates with even parity, a check is made on each group of bits received to make sure the total number of 1s in that group is even. If there is an odd number of 1s, an error has occurred.

As an illustration of how parity bits are attached to a code, Table 2–10 lists the parity bits for each BCD number for both even and odd parity. The parity bit for each BCD number is in the *P* column.

TABLE 2–10

EVEN PARITY		ODD PARITY	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

The parity bit can be attached to the code at either the beginning or the end, depending on system design. Notice that the total number of 1s, including the parity bit, is always even for even parity and always odd for odd parity.

Detecting an Error A parity bit provides for the detection of a single-bit error (or any odd number of errors, which is very unlikely) but cannot check for two errors in one group. For instance, let us assume that we wish to transmit the BCD code 0101. (Parity can be used with any number of bits; we are using four for illustration.) The total code transmitted, including the even parity bit, is

Even parity bit
00101
BCD code

Now, let us assume that an error occurs in the third bit from the left (the 1 becomes a 0) as follows:

Even parity bit
00001
Bit error

When this code is received, the parity check circuitry determines that there is only a single 1 (odd number), when there should be an even number of 1s. Because an even number of 1s does not appear in the code when it is received, an error is indicated.

An odd parity bit also provides in a similar manner for the detection of a single error in a given group of bits.

EXAMPLE 2–39

Assign the proper even parity bit to the following code groups:

(a) 1010 **(b)** 111000 **(c)** 101101

(d) 1000111001001 **(e)** 101101011111

Solution Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) 01010 **(b)** 1111000 **(c)** 0101101

(d) 0100011100101 **(e)** 1101101011111

Supplementary Problem Add an even parity bit to the 7-bit ASCII code for the letter K.

EXAMPLE 2–40

An odd parity system receives the following code groups: 10110, 11010, 110011, 110101110100, and 1100010101010. Determine which groups, if any, are in error.

Solution Since odd parity is required, any group with an even number of 1s is incorrect. The following groups are in error: **110011** and **1100010101010**.

Supplementary Problem The following ASCII character is received by an odd parity system: 00110111. Is it correct?

Error-Detection Codes

The *2-out-of-5 code* is sometimes used in communications work. It utilizes five bits to represent the ten decimal digits, so it is a form of BCD code. Each code word has exactly two 1s, a convention that facilitates decoding and provides for better error detection than the single-parity-bit method. If more or less than two 1s appear, an error is indicated.

The *63210 BCD code* is also characterized by having exactly two 1s in each of the 5-bit groups. Like the 2-out-of-5 code, it provides reliable error detection and is used in some applications.

The *biquinary* (two-five) *code* is used in certain counters and is composed of a 2-bit group and a 5-bit group, each with a single 1. Its weights are 50 43210. The 2-bit group, having weights of five and zero, indicates whether the number represented is less than, equal to, or greater than 5. The 5-bit group indicates the count above or below 5.

The *ring counter code* has ten bits, one for each decimal digit, and a single 1 makes error detection possible. It is easy to decode but wastes bits and requires more circuitry to implement than the 4-bit or 5-bit codes. The name is derived from the fact that the code is generated by a certain type of shift register, a ring counter. Its weights are 9876543210.

Each of these codes is listed in Table 2–11. You should realize that this is not an exhaustive coverage of all codes but simply an introduction to some of them.

▶ TABLE 2–11

DECIMAL	2-OUT-OF-5	63210	50 43210	9876543210
0	00011	00110	01 00001	0000000001
1	00101	00011	01 00010	0000000010
2	00110	00101	01 00100	0000000100
3	01001	01001	01 01000	0000001000
4	01010	01010	01 10000	0000010000
5	01100	01100	10 00001	0000100000
6	10001	10001	10 00010	0001000000
7	10010	10010	10 00100	0010000000
8	10100	10100	10 01000	0100000000
9	11000	11000	10 10000	1000000000

Hamming Error-Correction Code

This section discusses a method, generally known as the *Hamming code*, that not only provides for the detection of a bit error but also identifies the bit that is in error so that it can be corrected. The code uses a number of parity bits (dependent on the number of information bits), located at certain positions in the code group.

The Hamming code construction that follows is for single-error correction.

Number of Parity Bits

If the number of information bits is designated m, then the number of parity bits, p, is determined by the following relationship:

Equation 2–1

$$2^p \geq m + p + 1$$

For example, if we have four information bits, then p is found by trial and error with Equation 2–1. Let $p = 2$. Then,

$$2^p = 2^2 = 4$$

and,

$$m + p + 1 = 4 + 2 + 1 = 7$$

Since 2^p must be equal to or greater than $m + p + 1$, the relationship in Equation 2–1 is *not* satisfied. We have to try again. Let $p = 3$. Then,

$$2^p = 2^3 = 8$$

and,

$$m + p + 1 = 4 + 3 + 1 = 8$$

This value of p satisfies the relationship of Equation 2–1, so three parity bits are required to provide single-error correction for four information bits. It should be noted here that error detection and correction are provided for *all* bits, both parity and information, in a code group.

Placement of the Parity Bits in the Code

Now that we have found the number of parity bits required in our particular example, we must arrange the bits properly in the code. At this point you should realize that in this example, the code is composed of the four information bits and the three parity bits. The left-most bit is designated *bit 1*, the next bit is *bit 2*, and so on as follows:

bit 1, bit 2, bit 3 bit 4, bit 5, bit 6, bit 7

The parity bits are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8, ...), as indicated:

$$P_1,\ P_2,\ M_1,\ P_3,\ M_2,\ M_3,\ M_4$$

The symbol P_n designates a particular parity bit, and M_n designates a particular information bit.

Assignment of Parity Bit Values

Finally, we must properly assign a 1 or 0 value to each parity bit. Since each parity bit provides a check on certain other bits in the total code, we must know the value of these others in order to assign the parity bit value. To find the bit values, first number each bit position in binary, that is, write the binary number for each decimal position number (as shown in the second two rows of Table 2–12). Next, indicate the parity and information bit locations, as shown in the first row of Table 2–12. Notice that the binary position number of parity bit P_1 has a 1 for its right-most digit. *This parity bit checks all bit positions, including itself, that have 1s in the same location in the binary position numbers.* Therefore, parity bit P_1 checks bit positions 1, 3, 5, and 7.

The binary position number for parity bit P_2 has a 1 for its middle bit. It checks all bit positions, including itself, that have 1s in this same position. Therefore, parity bit P_2 checks bit positions 2, 3, 6, and 7.

▼TABLE 2-12

BIT DESIGNATION	P_1	P_2	M_1	P_3	M_2	M_3	M_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Information bits (M_n)							
Parity bits (P_n)							

The binary position number for parity bit P_3 has a 1 for its left-most bit. It checks all bit positions, including itself, that have 1s in this same position. Therefore, parity bit P_3 checks bit positions 4, 5, 6, and 7.

In each case, the parity bit is assigned a value to make the quantity of 1s in the set of bits that it checks odd or even, depending on which is specified. The following examples should make this procedure clear.

EXAMPLE 2–41

Determine the single-error-correcting code for the BCD number 1001 (information bits), using even parity.

Solution **Step 1.** Find the number of parity bits required. Let $p = 3$. Then,

$$2^p = 2^3 = 8$$

$$m + p + 1 = 4 + 3 + 1 = 8$$

Three parity bits are sufficient.

$$\text{Total code bits} = 4 + 3 = 7.$$

Step 2. Construct a bit position table, and enter the information bits. Parity bits are determined in the following steps.

Step 3. Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an even number of 1s (2) in this group.

Step 4. These parity bits are entered in the table, and the resulting combined code is 0011001. Table 2-13 shows the complete table.

BIT DESIGNATION	P_1	P_2	M_1	P_3	M_2	M_3	M_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Information bits			1		0	0	1
Parity bits	0	0		1			

▲TABLE 2-13

Supplementary Problem Determine single-error-correcting code for the BCD number 0110 using even parity.

EXAMPLE 2–42

Determine the single-error-correcting code for the information code 10110 for odd parity.

Solution **Step 1.** Determine the number of parity bits required. In this case the number of information bits, m, is five. From the previous example, we know that $p = 3$ will not work. Try $p = 4$:

$$2^p = 2^4 = 16$$

$$m + p + 1 = 5 + 4 + 1 = 10$$

Four parity bits are sufficient.

$$\text{Total code bits} = 5 + 4 = 9.$$

Step 2. Construct a bit position table, and enter the information bits. Parity bits are determined in the following steps.

Step 3. Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, 7, and 9 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an odd number of 1s (3) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_4 checks bit positions 8 and 9 and must be a 1 for there to be an odd number of 1s (1) in this group.

Step 4. These parity bits are entered in the table, and the resulting combined code is 101101110.

Table 2-14 shows the complete table.

BIT DESIGNATION	P_1	P_2	M_1	P_3	M_2	M_3	M_4	P_4	M_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Information bits			1		0	1	1		0
Parity bits	1	0		1				1	

TABLE 2–14

Supplementary Problem Determine single-error-correcting code for the information code 11010 for odd parity.

Detecting and Correcting an Error

Now that a method for constructing an error-correcting code has been covered, how do we use it to locate and correct an error? Each parity bit, along with its corresponding group of bits, must be checked for the proper parity. If there are three parity bits in a code word, then three parity checks are made. If there are four parity bits, four checks must be made, and so on. Each parity check will yield a good or a bad result. The total result of all the parity checks indicates the bit, if any, that is in error, as follows:

Step 1. Start with the group checked by P_1.

Step 2. Check the group for proper parity. A 0 represents a good parity check, and 1 represents a bad check.

Step 3. Repeat step 2 for each parity group.

Step 4. The binary number formed by the results of all the parity checks designates the position of the code bit that is in error. This is the *error position code*. The first parity check generates the least significant bit (LSB). If all checks are good, there is no error.

EXAMPLE 2–43

Assume that the code word in Example 2–41 (0011001) is transmitted and that 0010001 is received. The receiver does not "know" what was transmitted and must look for proper parities to determine if the code is correct. Designate any error that has occurred in transmission if even parity is used.

Solution First, make a bit position table:

TABLE 2–15

BIT DESIGNATION	P_1	P_2	M_1	P_3	M_2	M_3	M_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Received code	0	0	1	0	0	0	1

First parity check:
Bit P_1 checks positions 1, 3, 5, and 7.
There are two 1s in this group.
Parity check is good. → 0 (LSB)

Second parity check:
Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is good. → 0

Third parity check:
Bit P_3 checks positions 4, 5, 6, and 7.
There is one 1 in this group.
Parity check is bad. → 1 (MSB)

Result:
The error position code is 100 (binary four). This says that the bit in position 4 is in error. It is a 0 and should be a 1. The corrected code is 0011001, which agrees with the transmitted code.

Supplementary Problem If the Hamming code sequence 1100 110 is transmitted and due to error in one bit position, is received as 1110110, locate the position of error, assuming even parity.

EXAMPLE 2–44

The code 101101010 is received. Correct any errors. There are four parity bits, and odd parity is used.

Solution First, make a bit position table:

BIT DESIGNATION	P_1	P_2	M_1	P_3	M_2	M_3	M_4	P_4	M_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Received code	1	0	1	1	0	1	0	1	0

▲TABLE 2–16

First parity check:
Bit P_1 checks positions 1, 3, 5, 7, and 9.
There are two 1s in this group.
Parity check is bad. ——→ 1 (LSB)

Second parity check:
Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is bad. ——→ 1

Third parity check:
Bit P_3 checks positions 4, 5, 6, and 7.
There are two 1s in this group.
Parity check is bad. ——→ 1

Fourth parity check:
Bit P_4 checks positions 8 and 9.
There is one 1 in this group.
Parity check is good. ——→ 0 (MSB)

Result:
The error position code is 0111 (binary seven). This says that the bit in position 7 is in error. The corrected code is therefore 101101110.

Supplementary Problem Correct any errors in the code 111101001 received with four parity bits, assuming odd parity transmission.

SECTION 2–12 REVIEW

1. **Add an even parity bit to the following ASCII codes:**
   (a) 1001011 **(b)** 1110010 **(c)** 0100100 **(d)** 0101011
2. **Determine parity of**
   (a) 01110010 **(b)** 10011101
3. **What is the number of parity bits required for the construction of Hamming code for the following number of information bits?**
   (a) 4 **(b)** 5 **(c)** 7 **(d)** 8

SUMMARY

- A binary number is a weighted number in which the weight of each whole number digit is a positive power of two and the weight of each fractional digit is a negative power of two. The whole number weights increase from right to left—from least significant digit to most significant.
- A binary number can be converted to a decimal number by summing the decimal values of the weights of all the 1s in the binary number.
- A decimal whole number can be converted to binary by using the sum-of-weights or the repeated division-by-2 method.
- A decimal fraction can be converted to binary by using the sum-of-weights or the repeated multiplication-by-2 method.
- The basic rules for binary addition are as follows:

  $0 + 0 = 0$
  $0 + 1 = 1$
  $1 + 0 = 1$
  $1 + 1 = 10$

- The basic rules for binary subtraction are as follows:

  $0 - 0 = 0$
  $1 - 1 = 0$
  $1 - 0 = 1$
  $10 - 1 = 1$

- The 1's complement of a binary number is derived by changing 1s to 0s and 0s to 1s.
- The 2's complement of a binary number can be derived by adding 1 to the 1's complement.
- Binary subtraction can be accomplished with addition by using the 1's or 2's complement method.
- A positive binary number is represented by a 0 sign bit.
- A negative binary number is represented by a 1 sign bit.
- For arithmetic operations, negative binary numbers are represented in 1's complement or 2's complement form.
- In an addition operation, an overflow is possible when both numbers are positive or when both numbers are negative. An incorrect sign bit in the sum indicates the occurrence of an overflow.
- The hexadecimal number system consists of 16 digits and characters, 0 through 9 followed by A through F.
- One hexadecimal digit represents a 4-bit binary number, and its primary usefulness is in simplifying bit patterns and making them easier to read.
- A decimal number can be converted to hexadecimal by the repeated division-by-16 method.
- The octal number system consists of eight digits, 0 through 7.
- A decimal number can be converted to octal by using the repeated division-by-8 method.
- Octal-to-binary conversion is accomplished by simply replacing each octal digit with its 3-bit binary equivalent. The process is reversed for binary-to-octal conversion.
- A decimal number is converted to BCD by replacing each decimal digit with the appropriate 4-bit binary code.
- The ASCII is a 7-bit alphanumeric code that is widely used in computer systems for input and output of information.
- Parity bit is added to a group of bits to make number of 1s odd (odd parity) or even (even parity).
- Hamming code is used for detection and correction of error in digital transmission.

SELF-TEST

Answers are at the end of the chapter.

1. $2 \times 10^1 + 8 \times 10^0$ is equal to

   (a) 10 **(b)** 280 **(c)** 2.8 **(d)** 28

2. The binary number 1101 is equal to the decimal number

   (a) 13 **(b)** 49 **(c)** 11 **(d)** 3

3. The binary number 11011101 is equal to the decimal number

   (a) 121 **(b)** 221 **(c)** 441 **(d)** 256

4. The decimal number 17 is equal to the binary number

   (a) 10010 **(b)** 11000 **(c)** 10001 **(d)** 01001

5. The decimal number 175 is equal to the binary number

   (a) 11001111 **(b)** 10101110 **(c)** 10101111 **(d)** 11101111

6. The sum of 11010 + 01111 equals

   (a) 101001 **(b)** 101010 **(c)** 110101 **(d)** 101000

7. The difference of 110 − 010 equals

   (a) 001 **(b)** 010 **(c)** 101 **(d)** 100

8. The 1's complement of 10111001 is

   (a) 01000111 **(b)** 01000110 **(c)** 11000110 **(d)** 10101010

9. The 2's complement of 11001000 is

   (a) 00110111 **(b)** 00110001 **(c)** 01001000 **(d)** 00111000

10. The decimal number +122 is expressed in the 2's complement form as

    (a) 01111010 **(b)** 11111010 **(c)** 01000101 **(d)** 10000101

11. The decimal number −34 is expressed in the 2's complement form as

    (a) 01011110 **(b)** 10100010 **(c)** 11011110 **(d)** 01011101

12. A single-precision floating-point binary number has a total of

    (a) 8 bits **(b)** 16 bits **(c)** 24 bits **(d)** 32 bits

13. In the 2's complement form, the binary number 10010011 is equal to the decimal number

    (a) −19 **(b)** +109 **(c)** +91 **(d)** −109

14. The binary number 101100111001010100001 can be written in octal as

    (a) 5471230_8 **(b)** 5471241_8 **(c)** 2634521_8 **(d)** 23162501_8

15. The binary number 10001101010001101111 can be written in hexadecimal as

    (a) $AD467_{16}$ **(b)** $8C46F_{16}$ **(c)** $8D46F_{16}$ **(d)** $AE46F_{16}$

16. The binary number for $F7A9_{16}$ is

    (a) 1111011110101001 **(b)** 1110111110101001
    (c) 1111111010110001 **(d)** 1111011010101001

17. The BCD number for decimal 473 is

    (a) 111011010 **(b)** 110001110011 **(c)** 010001110011 **(d)** 010011110011

18. Refer to Table 2–7. The word STOP in ASCII is

    (a) 1010011101010010011111010000 **(b)** 1010010100110010011101010000
    (c) 1001010110110110011101010001 **(d)** 1010011101010010011101100100

19. The number of parity bits to be added to an 8-bit word for constructing Hamming code for detection and correction of single error is

    (a) 1 **(b)** 2 **(c)** 3 **(d)** 4

20. A 7-bit Hamming code (even parity) 0001001 for a BCD digit is known to have single error the encoded BCD digit is

    (a) 9 **(b)** 5 **(c)** 3 **(d)** 0

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 2–1 Decimal Numbers

1. What is the weight of the digit 6 in each of the following decimal numbers?

(a) 1386 **(b)** 54,692 **(c)** 671,920

2. Express each of the following decimal numbers as a power of ten:

(a) 10 **(b)** 100 **(c)** 10,000 **(d)** 1,000,000

3. Give the value of each digit in the following decimal numbers:

(a) 471 **(b)** 9356 **(c)** 125,000

4. How high can you count with four decimal digits?

SECTION 2–2 Binary Numbers

5. Convert the following binary numbers to decimal:

(a) 11 **(b)** 100 **(c)** 111 **(d)** 1000
(e) 1001 **(f)** 1100 **(g)** 1011 **(h)** 1111

6. Convert the following binary numbers to decimal:

(a) 1110 **(b)** 1010 **(c)** 11100 **(d)** 10000
(e) 10101 **(f)** 11101 **(g)** 10111 **(h)** 11111

7. Convert each binary number to decimal:

(a) 110011.11 **(b)** 101010.01 **(c)** 1000001.111
(d) 1111000.101 **(e)** 1011100.10101 **(f)** 1110001.0001
(g) 1011010.1010 **(h)** 1111111.11111

8. What is the highest decimal number that can be represented by each of the following numbers of binary digits (bits)?

(a) two **(b)** three **(c)** four **(d)** five **(e)** six
(f) seven **(g)** eight **(h)** nine **(i)** ten **(j)** eleven

9. How many bits are required to represent the following decimal numbers?

(a) 17 **(b)** 35 **(c)** 49 **(d)** 68
(e) 81 **(f)** 114 **(g)** 132 **(h)** 205

10. Generate the binary sequence for each decimal sequence:

(a) 0 through 7 **(b)** 8 through 15 **(c)** 16 through 31
(d) 32 through 63 **(e)** 64 through 75

SECTION 2–3 Decimal-to-Binary Conversion

11. Convert each decimal number to binary by using the sum-of-weights method:

(a) 10 **(b)** 17 **(c)** 24 **(d)** 48
(e) 61 **(f)** 93 **(g)** 125 **(h)** 186

12. Convert each decimal fraction to binary using the sum-of-weights method:

(a) 0.32 **(b)** 0.246 **(c)** 0.0981

13. Convert each decimal number to binary using repeated division by 2:

(a) 15 **(b)** 21 **(c)** 28 **(d)** 34
(e) 40 **(f)** 59 **(g)** 65 **(h)** 73

14. Convert each decimal fraction to binary using repeated multiplication by 2:

(a) 0.98 **(b)** 0.347 **(c)** 0.9028

SECTION 2–4 Binary Arithmetic

15. Add the binary numbers:

(a) 11 + 01 **(b)** 10 + 10 **(c)** 101 + 11

(d) 111 + 110 **(e)** 1001 + 101 **(f)** 1101 + 1011

16. Use direct subtraction on the following binary numbers:

(a) 11 − 1 **(b)** 101 − 100 **(c)** 110 − 101

(d) 1110 − 11 **(e)** 1100 − 1001 **(f)** 11010 − 10111

17. Perform the following binary multiplications:

(a) 11 × 11 **(b)** 100 × 10 **(c)** 111 × 101

(d) 1001 × 110 **(e)** 1101 × 1101 **(f)** 1110 × 1101

18. Divide the binary numbers as indicated:

(a) 100 ÷ 10 **(b)** 1001 ÷ 11 **(c)** 1100 ÷ 100

SECTION 2–5 1's and 2's Complements of Binary Numbers

19. Determine the 1's complement of each binary number:

(a) 101 **(b)** 110 **(c)** 1010

(d) 11010111 **(e)** 1110101 **(f)** 00001

20. Determine the 2's complement of each binary number using either method:

(a) 10 **(b)** 111 **(c)** 1001 **(d)** 1101

(e) 11100 **(f)** 10011 **(g)** 10110000 **(h)** 00111101

SECTION 2–6 Signed Numbers

21. Express each decimal number in binary as an 8-bit sign-magnitude number:

(a) +29 **(b)** −85 **(c)** +100 **(d)** −123

22. Express each decimal number as an 8-bit number in the 1's complement form:

(a) −34 **(b)** +57 **(c)** −99 **(d)** +115

23. Express each decimal number as an 8-bit number in the 2's complement form:

(a) +12 **(b)** −68 **(c)** +101 **(d)** −125

24. Determine the decimal value of each signed binary number in the sign-magnitude form:

(a) 10011001 **(b)** 01110100 **(c)** 10111111

25. Determine the decimal value of each signed binary number in the 1's complement form:

(a) 10011001 **(b)** 01110100 **(c)** 10111111

26. Determine the decimal value of each signed binary number in the 2's complement form:

(a) 10011001 **(b)** 01110100 **(c)** 10111111

27. Express each of the following sign-magnitude binary numbers in single-precision floating-point format:

(a) 0111110000101011 **(b)** 100110000011000

28. Determine the values of the following single-precision floating-point numbers:

(a) 1 10000001 01001001110001000000000

(b) 0 11001100 10000111110100100000000

SECTION 2–7 Arithmetic Operations with Signed Numbers

29. Convert each pair of decimal numbers to binary and add using the 2's complement form:

(a) 33 and 15 **(b)** 56 and −27 **(c)** −46 and 25 **(d)** −110 and −84

30. Perform each addition in the 2's complement form:

(a) 00010110 + 00110011 **(b)** 01110000 + 10101111

31. Perform each addition in the 2's complement form:

(a) 10001100 + 00111001 **(b)** 11011001 + 11100111

32. Perform each subtraction in the 2's complement form:

(a) 00110011 − 00010000 **(b)** 01100101 − 11101000

33. Multiply 01101010 by 11110001 in the 2's complement form.

34. Divide 01000100 by 00011001 in the 2's complement form.

SECTION 2-8 Hexadecimal Numbers

35. Convert each hexadecimal number to binary:

(a) 38_{16} **(b)** 59_{16} **(c)** $A14_{16}$ **(d)** $5C8_{16}$

(e) 4100_{16} **(f)** $FB17_{16}$ **(g)** $8A9D_{16}$

36. Convert each binary number to hexadecimal:

(a) 1110 **(b)** 10 **(c)** 10111

(d) 10100110 **(e)** 1111110000 **(f)** 100110000010

37. Convert each hexadecimal number to decimal:

(a) 23_{16} **(b)** 92_{16} **(c)** $1A_{16}$ **(d)** $8D_{16}$

(e) $F3_{16}$ **(f)** EB_{16} **(g)** $5C2_{16}$ **(h)** 700_{16}

38. Convert each decimal number to hexadecimal:

(a) 8 **(b)** 14 **(c)** 33 **(d)** 52

(e) 284 **(f)** 2890 **(g)** 4019 **(h)** 6500

39. Perform the following additions:

(a) $37_{16} + 29_{16}$ **(b)** $A0_{16} + 6B_{16}$ **(c)** $FF_{16} + BB_{16}$

40. Perform the following subtractions:

(a) $51_{16} - 40_{16}$ **(b)** $C8_{16} - 3A_{16}$ **(c)** $FD_{16} - 88_{16}$

SECTION 2-9 Octal Numbers

41. Convert each octal number to decimal:

(a) 12_8 **(b)** 27_8 **(c)** 56_8 **(d)** 64_8 **(e)** 103_8

(f) 557_8 **(g)** 163_8 **(h)** 1024_8 **(i)** 7765_8

42. Convert each decimal number to octal by repeated division by 8:

(a) 15 **(b)** 27 **(c)** 46 **(d)** 70

(e) 100 **(f)** 142 **(g)** 219 **(h)** 435

43. Convert each octal number to binary:

(a) 13_8 **(b)** 57_8 **(c)** 101_8 **(d)** 321_8 **(e)** 540_8

(f) 4653_8 **(g)** 13271_8 **(h)** 45600_8 **(i)** 100213_8

44. Convert each binary number to octal:

(a) 111 **(b)** 10 **(c)** 110111

(d) 101010 **(e)** 1100 **(f)** 1011110

(g) 101100011001 **(h)** 10110000011 **(i)** 111111101111000

SECTION 2-10 Binary Coded Decimal (BCD)

45. Convert each of the following decimal numbers to 8421 BCD:

(a) 10 **(b)** 13 **(c)** 18 **(d)** 21 **(e)** 25 **(f)** 36

(g) 44 **(h)** 57 **(i)** 69 **(j)** 98 **(k)** 125 **(l)** 156

46. Convert each of the decimal numbers in Problem 45 to straight binary, and compare the number of bits required with that required for BCD.

47. Convert the following decimal numbers to BCD:

    (a) 104 (b) 128 (c) 132 (d) 150 (e) 186
    (f) 210 (g) 359 (h) 547 (i) 1051

48. Convert each of the BCD numbers to decimal:

    (a) 0001 (b) 0110 (c) 1001
    (d) 00011000 (e) 00011001 (f) 00110010
    (g) 01000101 (h) 10011000 (i) 100001110000

49. Convert each of the BCD numbers to decimal:

    (a) 10000000 (b) 001000110111
    (c) 001101000110 (d) 010000100001
    (e) 011101010100 (f) 100000000000
    (g) 100101111000 (h) 0001011010000011
    (i) 1001000000011000 (j) 0110011001100111

50. Add the following BCD numbers:

    (a) 0010 + 0001 (b) 0101 + 0011
    (c) 0111 + 0010 (d) 1000 + 0001
    (e) 00011000 + 00010001 (f) 01100100 + 00110011
    (g) 01000000 + 01000111 (h) 10000101 + 00010011

51. Add the following BCD numbers:

    (a) 1000 + 0110 (b) 0111 + 0101
    (c) 1001 + 1000 (d) 1001 + 0111
    (e) 00100101 + 00100111 (f) 01010001 + 01011000
    (g) 10011000 + 10010111 (h) 010101100001 + 011100001000

52. Convert each pair of decimal numbers to BCD, and add as indicated:

    (a) 4 + 3 (b) 5 + 2 (c) 6 + 4 (d) 17 + 12
    (e) 28 + 23 (f) 65 + 58 (g) 113 + 101 (h) 295 + 157

SECTION 2–11 Digital Codes

53. In a certain application a 4-bit binary sequence cycles from 1111 to 0000 periodically. There are four-bit changes, and because of circuit delays, these changes may not occur at the same instant. For example, if the LSB changes first, the number will appear as 1110 during the transition from 1111 to 0000 and may be misinterpreted by the system. Illustrate how the Gray code avoids this problem.

54. Convert each binary number to Gray code:

    (a) 11011 (b) 1001010 (c) 1111011101110

55. Convert each Gray code to binary:

    (a) 1010 (b) 00010 (c) 11000010001

56. Convert each of the following decimal numbers to ASCII. Refer to Table 2–7.

    (a) 1 (b) 3 (c) 6 (d) 10 (e) 18
    (f) 29 (g) 56 (h) 75 (i) 107

57. Determine each ASCII character. Refer to Table 2–7.

    (a) 0011000 (b) 1001010 (c) 0111101
    (d) 0100011 (e) 0111110 (f) 1000010

58. Decode the following ASCII coded message:

    1001000 1100101 1101100 1101100 1101111 0101110
    0100000 1001000 1101111 1110111 0100000 1100001
    1110010 1100101 0100000 1111001 1101111 1110101
    0111111

59. Write the message in Problem 58 in hexadecimal.
60. Convert the following computer program statement to ASCII:

```
30 INPUT A, B
```

SECTION 2–12 Error-Detection and Correction Codes

61. Determine which of the following even parity codes are in error:
    (a) 100110010 (b) 011101010 (c) 10111111010001010
62. Determine which of the following odd parity codes are in error:
    (a) 11110110 (b) 00110001 (c) 01010101010101010
63. Attach the proper even parity bit to each of the following bytes of data:
    (a) 10100100 (b) 00001001 (c) 11111110
64. Attach the proper odd parity bit to each of the bytes of Problem 63.
65. Determine the single-error-correcting code for the following BCD numbers, using odd parity:
    (a) 0000 (b) 0001 (c) 0010 (d) 1000
66. 8421 codes are transmitted as Hamming codes with even parity and the following words are received:
    (a) 0101000 (b) 0011101 (c) 1100100 (d) 1101001
    Find the words that have single error and determine the correct BCD digit.

ANSWERS

SECTION REVIEWS

SECTION 2–1 Decimal Numbers

1. **(a)** 1370: 10 **(b)** 6725: 100 **(c)** 7051: 1000 **(d)** 58.72: 0.1
2. **(a)** $51 = (5 \times 10) + (1 \times 1)$ **(b)** $137 = (1 \times 100) + (3 \times 10) + (7 \times 1)$
   (c) $1492 = (1 \times 1000) + (4 \times 100) + (9 \times 10) + (2 \times 1)$
   (d) $106.58 = (1 \times 100) + (0 \times 10) + (6 \times 1) + (5 \times 0.1) + (8 \times 0.01)$

SECTION 2–2 Binary Numbers

1. $2^8 - 1 = 255$
2. Weight is 16.
3. 10111101.011 = 189.375

SECTION 2–3 Decimal-to-Binary Conversion

1. **(a)** 23 = 10111 **(b)** 57 = 111001 **(c)** 45.5 = 101101.1
2. **(a)** 14 = 1110 **(b)** 21 = 10101 **(c)** 0.375 = 0.011

SECTION 2–4 Binary Arithmetic

1. **(a)** 1101 + 1010 = 10111 **(b)** 10111 + 01101 = 100100
2. **(a)** 1101 − 0100 = 1001 **(b)** 1001 − 0111 = 0010
3. **(a)** 110 × 111 = 101010 **(b)** 1100 ÷ 011 = 100

SECTION 2–5 1's and 2's Complements of Binary Numbers

1. **(a)** 1's comp of 00011010 = 11100101 **(b)** 1's comp of 11110111 = 00001000
   (c) 1's comp of 10001101 = 01110010

2. (a) 2's comp of 00010110 = 11101010 (b) 2's comp of 11111100 = 00000100
 (c) 2's comp of 10010001 = 01101111

SECTION 2–6 Signed Numbers

1. Sign-magnitude: +9 = 00001001
2. 1's comp: −33 = 11011110
3. 2's comp: −46 = 11010010
4. Exponent and mantissa

SECTION 2–7 Arithmetic Operations with Signed Numbers

1. Cases of addition: positive number is larger, negative number is larger, both are positive, both are negative
2. 00100001 + 10111100 = 11011101
3. 01110111 − 00110010 = 01000101
4. Sign of product is positive.
5. 00000101 × 01111111 = 01001111011
6. Sign of quotient is negative.
7. 00110000 ÷ 00001100 = 00000100

SECTION 2–8 Hexadecimal Numbers

1. (a) 10110011 = $B3_{16}$ (b) 110011101000 = $CE8_{16}$
2. (a) 57_{16} = 01010111 (b) $3A5_{16}$ = 001110100101
 (c) $F8OB_{16}$ = 1111100000001011
3. $9B30_{16} = 39{,}728_{10}$
4. $573_{10} = 23D_{16}$
5. (a) $18_{16} + 34_{16} = 4C_{16}$ (b) $3F_{16} + 2A_{16} = 69_{16}$
6. $75_{16} - 21_{16} = 54_{16}$ (b) $94_{16} - 5C_{16} = 38_{16}$

SECTION 2–9 Octal Numbers

1. (a) $73_8 = 59_{10}$ (b) $125_8 = 85_{10}$
2. (a) $98_{10} = 142_8$ (b) $163_{10} = 243_8$
3. (a) 46_8 = 100110 (b) 723_8 = 111010011 (c) 5624_8 = 101110010100
4. (a) 110101111 = 657_8 (b) 1001100010 = 1142_8 (c) 10111111001 = 2771_8

SECTION 2–10 Binary Coded Decimal (BCD)

1. (a) 0010: 2 (b) 1000: 8 (c) 0001: 1 (d) 0100: 4
2. (a) 6_{10} = 0110 (b) 15_{10} = 00010101 (c) 273_{10} = 001001110011
 (d) 849_{10} = 100001001001
3. (a) 10001001 = 89_{10} (b) 001001111000 = 278_{10} (c) 000101010111 = 157_{10}
4. A 4-bit sum is invalid when it is greater than 9_{10}.

SECTION 2–11 Digital Codes

1. **(a)** 1100_2 = 1010 Gray **(b)** 1010_2 = 1111 Gray **(c)** 11010_2 = 10111 Gray

2. **(a)** 1000 Gray = 1111_2 **(b)** 1010 Gray = 1100_2 **(c)** 11101 Gray = 10110_2

3. **(a)** K: 1001011 → $4B_{16}$ **(b)** r: 1110010 → 72_{16}
(c) \$: 0100100 → 24_{16} **(d)** +: 0101011 → $2B_{16}$

SECTION 2–12 Error-Detection and Correction Codes

1. **(a)** *0*1001011 **(b)** *0*1110010 **(c)** *0*0100100 **(d)** *0*0101011

2. **(a)** Even **(b)** odd

3. **(a)** 3 **(b)** 4 **(c)** 4 **(d)** 4

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

2–1 9 has a value of 900, 3 has a value of 30, 9 has a value of 9.

2–2 6 has a value of 60, 7 has a value of 7, 9 has a value of 9/10 (0.9), 2 has a value of 2/100 (0.02), 4 has a value of 4/1000 (0.004).

2–3 10010001 = 128 + 16 + 1 = 145 **2–4** 10.111 = 2 + 0.5 + 0.25 + 0.125 = 2.875

2–5 125 = 64 + 32 + 16 + 8 + 4 + 1 = 1111101 **2–6** 39 = 100111

2–7 1111 + 1100 = 11011 **2–8** 111 − 100 = 011 **2–9** 110 − 101 = 001

2–10 1101 × 1010 = 10000010 **2–11** 1100 ÷ 100 = 11 **2–12** 00110101

2–13 01000000 **2–14** See Table 2–17. **2–15** 01110111 = $+119_{10}$

▶ TABLE 2–17

	SIGN-MAGNITUDE	1'S COMP	2'S COMP
+19	00010011	00010011	00010011
−19	10010011	11101100	11101101

2–16 11101011 = -20_{10} **2–17** 11010111 = -41_{10}

2–18 11000010001010011000000000 **2–19** 01010101 **2–20** 00010001

2–21 1001000110 **2–22** (83)(−59) = −4897 (10110011011111 in 2's comp)

2–23 100 ÷ 25 = 4 (0100) **2–24** $4F79C_{16}$ **2–25** 0110101111010011_2

2–26 $6BD_{16} = 011010111101 = 2^{10} + 2^9 + 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0$
$= 1024 + 512 + 128 + 32 + 16 + 8 + 4 + 1 = 1725_{10}$

2–27 $60A_{16} = (6 \times 256) + (0 \times 16) + (10 \times 1) = 1546_{10}$

2–28 $2591_{10} = A1F_{16}$ **2–29** $4C_{16} + 3A_{16} = 86_{16}$ **2–30** $BCD_{16} - 173_{16} = A5A_{16}$

2–31 **(a)** $001011_2 = 11_{10} = 13_8$ **(b)** $010101_2 = 21_{10} = 25_8$
(c) $001100000_2 = 96_{10} = 140_8$ **(d)** $111101010110_2 = 3926_{10} = 7526_8$

2–32 1250762_8 **2–33** 1001011001110011 **2–34** $82{,}276_{10}$

2–35 1001100101101000 **2–36** 10000010 **2–37** **(a)** 111011 (Gray) **(b)** 111010_2

2–38 The sequence of codes for `80 INPUT Y` is $38_{16}30_{16}20_{16}49_{16}4E_{16}50_{16}55_{16}54_{16}20_{16}59_{16}$

2–39 01001011 **2–40** Yes **2–41** 1100110 **2–42** 011110110 **2–43** 3

2–44 111001001

SELF-TEST

1. (d) **2.** (a) **3.** (b) **4.** (c) **5.** (c) **6.** (a) **7.** (d) **8.** (b)
9. (d) **10.** (a) **11.** (c) **12.** (d) **13.** (d) **14.** (b) **15.** (c) **16.** (a)
17. (c) **18.** (a) **19.** (d) **20.** (a)

3

LOGIC GATES

CHAPTER OBJECTIVES

- Describe the operation of the inverter, the AND gate, and the OR gate
- Describe the operation of the NAND gate and the NOR gate
- Express the operation of NOT, AND, OR, NAND, and NOR gates with Boolean algebra
- Describe the operation of the exclusive-OR and exclusive-NOR gates
- Recognize and use both the distinctive shape logic gate symbols and the rectangular outline logic gate symbols of ANSI/IEEE Standard 91–1984
- Construct timing diagrams showing the proper time relationships of inputs and outputs for the various logic gates
- List specific fixed-function integrated circuit devices that contain the various logic gates
- Use each logic gate in simple applications

INTRODUCTION

The basic logic operations have been discussed in Section 1–3. The emphasis in this chapter is on the operation and applications of logic gates. The relationship of input and output waveforms of a gate using timing diagrams is thoroughly covered.

Logic symbols used to represent the logic gates are in accordance with ANSI/IEEE Standard 91–1984. This standard has been adopted by private industry and the military for use in internal documentation as well as published literature.

Since integrated circuits (ICs) are used in all applications, the logic function of a device is generally of greater importance to the user rather than the details of the component-level circuit operation within the IC package. Therefore, for simplicity in the use of available ICs, only their pin connections have been covered. The component level circuit operation and other relevant details will be covered in Chapter 11 on Integrated Circuit Technologies.

3–1 THE INVERTER

The inverter (NOT circuit) performs the operation called ***inversion*** or *complementation*. The inverter changes one logic level to the opposite level. In terms of bits, it changes a 1 to a 0 and a 0 to a 1.

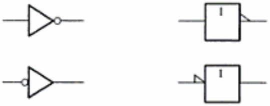
After completing this section, you should be able to

- Identify negation and polarity indicators
- Identify an inverter by either its distinctive shape symbol or its rectangular outline symbol
- Produce the truth table for an inverter
- Describe the logical operation of an inverter

Standard logic symbols for the **inverter** are shown in Figure 3–1. Part (a) shows the *distinctive shape* symbols, and part (b) shows the *rectangular outline* symbols. In this textbook, distinctive shape symbols are generally used; however, the rectangular outline symbols are found in many industry publications, and you should become familiar with them as well. (Logic symbols are in accordance with **ANSI/IEEE** Standard 91–1984.)

▶ **FIGURE 3–1**

Standard logic symbols for inverter



(a) Distinctive shape symbols with negation indicators

(b) Rectangular outline symbols with polarity indicators

The Negation and Polarity Indicators

The negation indicator is a "bubble" (○) that indicates **inversion** or *complementation* when it appears on the input or output of any logic element, as shown in Figure 3–1(a) for the inverter. Generally, inputs are on the left of a logic symbol and the output is on the right. When appearing on the input, the bubble means that a 0 is the active or *asserted* input state, and the input is called an active-LOW input. When appearing on the output, the bubble means that a 0 is the active or asserted output state, and the output is called an active-LOW output. The absence of a bubble on input or output means that a 1 is the active or asserted state, and in this case, the input or output is called active-HIGH.

The polarity or level indicator is a "triangle" (◺) that indicates inversion when it appears on the input or output of a logic element, as shown in Figure 3–1(b). When appearing on the input, it means that a LOW level is the active or asserted input state. When appearing on the output, it means that a LOW level is the active or asserted output state.

Either indicator (bubble or triangle) can be used both on distinctive shape symbols and on rectangular outline symbols. Figure 3–1(a) indicates the principal inverter symbols used in this text. Note that a change in the placement of the negation or polarity indicator does not imply a change in the way an inverter operates.

Inverter Truth Table

When a HIGH level is applied to an inverter input, a LOW level will appear on its output. When a LOW level is applied to its input, a HIGH will appear on its output. This operation is summarized in Table 3–1, which shows the output for each possible input in terms of levels and corresponding bits. A table such as this is called a **truth table.**

▶ **TABLE 3–1**

Inverter truth table

INPUT	OUTPUT
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)

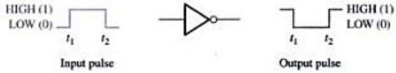
Inverter Operation

Figure 3–2 shows the output of an inverter for a pulse input, where t_1 and t_2 indicate the corresponding points on the input and output pulse waveforms.

When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW, thereby producing an inverted output pulse.

FIGURE 3–2

Inverter operation with a pulse input

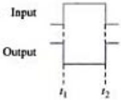


Timing Diagrams

Recall from Chapter 1 that a **timing diagram** is basically *a graph that accurately displays the relationship of two or more waveforms with respect to each other on a time basis.* For example, the time relationship of the output pulse to the input pulse in Figure 3–2 can be shown with a simple timing diagram by aligning the two pulses so that the occurrences of the pulse edges appear in the proper time relationship. The rising edge of the input pulse and the falling edge of the output pulse occur at the same time (ideally). Similarly, the falling edge of the input pulse and the rising edge of the output pulse occur at the same time (ideally). This timing relationship is shown in Figure 3–3. Timing diagrams are especially useful for illustrating the time relationship of digital waveforms with multiple pulses, as Example 3–1 illustrates.

FIGURE 3–3

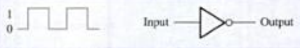
Timing diagram for the case in Figure 3–2



EXAMPLE 3–1

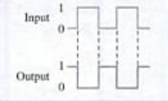
A waveform is applied to an inverter in Figure 3–4. Determine the output waveform corresponding to the input and show the timing diagram. According to the placement of the bubble, what is the active output state?

FIGURE 3–4



Solution The output waveform is exactly opposite to the input (inverted), as shown in Figure 3–5, which is the basic timing diagram. The active or asserted output state is **0**.

FIGURE 3–5



Supplementary Problem If the inverter is shown with the negative indicator (bubble) on the input instead of the output, how is the timing diagram affected

Logic Expression for an Inverter

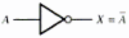
In **Boolean algebra**, which is the mathematics of logic circuits and will be covered thoroughly in Chapter 4, a variable is designated by a letter. The **complement** of a variable is designated by a bar over the letter. A variable can take on a value of either 1 or 0. If a given variable is 1, its complement is 0 and vice versa. Boolean algebra uses variables and operators to describe a logic circuit.

The operation of an inverter (NOT circuit) can be expressed as follows: If the input variable is called A and the output variable is called X, then

$$X = \overline{A}$$

This expression states that the output is the complement of the input, so if $A = 0$, then $X = 1$, and if $A = 1$, then $X = 0$. Figure 3–6 illustrates this. The complemented variable $\overline{A}$ can be read as "A bar" or "not A."

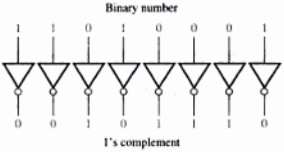
▶ **FIGURE 3–6**



Application Example

Figure 3–7 shows a circuit for producing the 1's complement of an 8-bit binary number. The bits of the binary number are applied to the inverter inputs and the 1's complement of the number appears on the outputs.

▶ **FIGURE 3–7**



SECTION 3–1 REVIEW

Answers are at the end of the chapter.

1. When a 1 is on the input of an inverter, what is the output?
2. An active HIGH pulse (HIGH level when asserted, LOW level when not) is required on an inverter input.
   (a) Draw the appropriate logic symbol, using the distinctive shape and the negation indicator, for the inverter in this application.
   (b) Describe the output when a positive-going pulse is applied to the input of an inverter.

3–2 THE AND GATE

The AND gate is one of the basic gates that can be combined to form any logic function. An AND gate can have two or more inputs and performs what is known as logical multiplication.

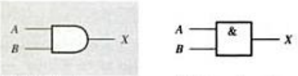
After completing this section, you should be able to

- Identify an AND gate by its distinctive shape symbol or by its rectangular outline symbol
- Describe the operation of an AND gate
- Generate the truth table for an AND gate with any number of inputs
- Produce a timing diagram for an AND gate with any specified input waveforms
- Write the logic expression for an AND gate with any number of inputs
- Discuss example of AND gate application

The term *gate* is used to describe a circuit that performs a basic logic operation. The AND gate is composed of two or more inputs and a single output, as indicated by the standard logic symbols shown in Figure 3–8. Inputs are on the left, and the output is on the right in each symbol. Gates with two inputs are shown; however, an AND gate can have any number of inputs greater than one. Although examples of both distinctive shape symbols and rectangular outline symbols are shown, the distinctive shape symbol, shown in part (a), is used predominantly in this book.

FIGURE 3–8

Standard logic symbols for the AND gate



(a) Distinctive shape

(b) Rectangular outline with the AND (&) qualifying symbol

Operation of an AND Gate

An AND gate produces a HIGH output *only* when *all* of the inputs are HIGH. When any of the inputs is LOW, the output is LOW. Therefore, the basic purpose of an AND gate is to determine when certain conditions are simultaneously true, as indicated by HIGH levels on all of its inputs, and to produce a HIGH on its output to indicate that all these conditions are true. The inputs of the 2-input AND gate in Figure 3–8 are labelled *A* and *B*, and the output is labelled *X*. The gate operation can be stated as follows:

> **For a 2-input AND gate, output *X* is HIGH if inputs *A* and *B* are HIGH; *X* is LOW if either *A* or *B* is LOW, or if both *A* and *B* are LOW.**

Figure 3–9 illustrates a 2-input AND gate with all four possibilities of input combinations and the resulting output for each.

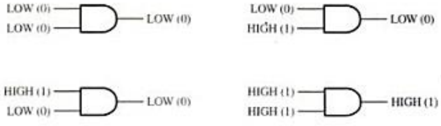


FIGURE 3–9

An AND gate can have more than two inputs

AND Gate Truth Table

The logical operation of a gate can be expressed with a truth table that lists all input combinations with the corresponding outputs, as illustrated in Table 3–2 for a 2-input AND gate. The truth table can be expanded to any number of inputs. Although the terms HIGH and LOW tend to give a "physical" sense to input and output states, the truth table is shown with 1s and 0s; a HIGH is equivalent to a 1 and a LOW is equivalent to a 0 in positive logic. For any AND gate, regardless of the number of inputs, the output is HIGH *only* when *all* inputs are HIGH.

▶ **TABLE 3–2**

Truth table for AND gate

INPUTS		OUTPUT
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

1 = HIGH, 0 = LOW

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

Equation 3–1

$$N = 2^n$$

where N is the number of possible input combinations and n is the number of input variables. To illustrate,

For two input variables: $N = 2^2 = 4$ combinations

For three input variables: $N = 2^3 = 8$ combinations

For four input variables: $N = 2^4 = 16$ combinations

You can determine the number of input bit combinations for gates with any number of inputs by using Equation 3–1.

EXAMPLE 3–2

(a) Develop the truth table for a 3-input AND gate.

(b) Determine the total number of possible input combinations for a 4-input AND gate.

Solution **(a)** There are eight possible input combinations ($2^3 = 8$) for a 3-input AND gate. The input side of the truth table (Table 3–3) shows all eight combinations of three bits. The output side is all 0s except when all three input bits are 1s.

▶ TABLE 3-3

INPUTS			OUTPUT
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b) $N = 2^4 = 16$. There are 16 possible combinations of input bits for a 4-input AND gate.

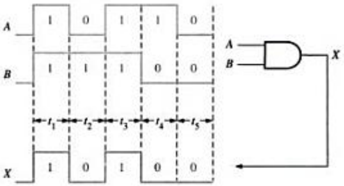
Supplementary Problem Develop the truth table for a 4-input AND gate.

Pulsed Operation

In most applications, the inputs to a gate are not stationary levels but are voltage waveforms that change frequently between HIGH and LOW logic levels. Now, let us look at the operation of AND gates with pulse waveform inputs, keeping in mind that an AND gate obeys the truth table operation regardless of whether its inputs are constant levels or levels that change back and forth.

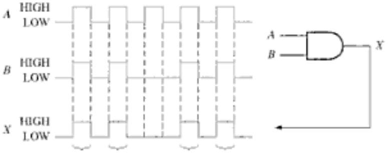
Let us examine the pulsed operation of an AND gate by looking at the inputs with respect to each other in order to determine the output level at any given time. In Figure 3–10, the inputs are both HIGH (1) during the time interval, t_1, making the output HIGH (1) during this interval. During time interval t_2, input A is LOW (0) and input B is HIGH (1), so the output is LOW (0). During time interval t_3, both inputs are HIGH (1) again, and therefore the output is HIGH (1). During time interval t_4, input A is HIGH (1) and input B is LOW (0), resulting in a LOW (0) output. Finally, during time interval t_5, input A is LOW (0), input B is LOW (0), and the output is therefore LOW (0). As you know, a diagram of input and output waveforms showing time relationships is called a *timing diagram*.

▶ FIGURE 3-10



EXAMPLE 3–3

If two waveforms, *A* and *B*, are applied to the AND gate inputs as in Figure 3–11, what is the resulting output waveform?



A and *B* are both HIGH during these four time intervals. Therefore *X* is HIGH.

▲ FIGURE 3–11

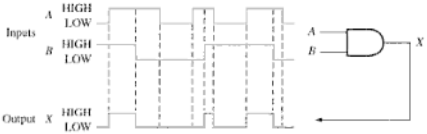
Solution The output waveform *X* is HIGH only when both *A* and *B* waveforms are HIGH as shown in the timing diagram in Figure 3–11.

Supplementary Problem Determine the output waveform and show a timing diagram if the second and fourth pulses in waveform *A* of Figure 3–11 are replaced by LOW levels.

Remember, when analyzing the pulsed operation of logic gates, it is important to pay careful attention to the time relationships of all the inputs with respect to each other and to the output.

EXAMPLE 3–4

For the two input waveforms, *A* and *B*, in Figure 3–12, show the output waveform with its proper relation to the inputs.



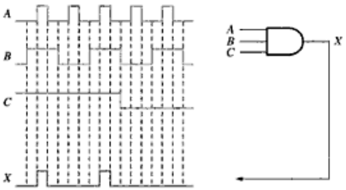
▲ FIGURE 3–12

Solution The output waveform is HIGH only when both of the input waveforms are HIGH as shown in the timing diagram.

Supplementary Problem Show the output waveform if the *B* input to the AND gate in Figure 3–12 is always HIGH.

EXAMPLE 3–5

For the 3-input AND gate in Figure 3–13, determine the output waveform in relation to the inputs.



▲ **FIGURE 3–13**

Solution The output waveform X of the 3-input AND gate is HIGH only when all three input waveforms A, B, and C are HIGH.

Supplementary Problem What is the output waveform of the AND gate in Figure 3–13 if the C input is always HIGH?

Logic Expressions for an AND Gate

The logical AND function of two variables is represented mathematically either by placing a dot between the two variables, as $A \cdot B$, or by simply writing the adjacent letters without the dot, as AB. We will normally use the latter notation because it is easier to write.

Boolean multiplication follows the same basic rules governing binary multiplication, which were discussed in Chapter 2 and are as follows:

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

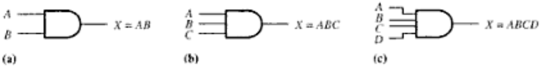
Boolean multiplication is the same as the AND function.

The operation of a 2-input AND gate can be expressed in equation form as follows: If one input variable is A, the other input variable is B, and the output variable is X, then the Boolean expression is

$$X = AB$$

Figure 3–14(a) shows the gate with the input and output variables indicated. When variables are shown together like AB, ABC, $ABCD$, they are ANDed.

► **FIGURE 3–14**



To extend the AND expression to more than two input variables, simply use a new letter for each input variable. The function of a 3-input AND gate, for example, can be expressed as $X = ABC$, where A, B, and C are the input variables. The expression for a 4-input AND gate can be $X = ABCD$, and so on. Parts (b) and (c) of Figure 3–14 show AND gates with three and four input variables, respectively.

You can evaluate an AND gate operation by using the Boolean expressions for the output. For example, each variable on the inputs can be either a 1 or a 0; so for the 2-input AND gate, make substitutions in the equation for the output, $X = AB$, as shown in Table 3–4. This evaluation shows that the output X of an AND gate is a 1 (HIGH) only when both inputs are 1s (HIGHs). A similar analysis can be made for any number of input variables.

TABLE 3–4

A	B	$AB = X$
0	0	$0 \cdot 0 = 0$
0	1	$0 \cdot 1 = 0$
1	0	$1 \cdot 0 = 0$
1	1	$1 \cdot 1 = 1$

Application Examples

The AND Gate as an Enable/Inhibit Device A common application of the AND gate is to **enable** (that is, to allow) the passage of a signal (pulse waveform) from one point to another at certain times and to inhibit (prevent) the passage at other times.

A simple example of this particular use of an AND gate is shown in Figure 3–15, where the AND gate controls the passage of a signal (waveform A). The enable pulse is applied at B input. When the enable pulse is HIGH, waveform A passes through the gate, and when the enable pulse is LOW, waveform A is prevented from passing through the gate, that is, inhibited (or the gate is disabled).

During the 1 second (1 s) interval of the enable pulse, a certain number of pulses in waveform A pass through the AND gate to the counter. The number of pulses passing through during the 1 s interval is equal to the frequency of waveform A. For example, Figure 3–15 shows six pulses in one second, which is a frequency of 6 Hz. If 1000 pulses pass through the gate in the 1 s interval of the enable pulse, there are 1000 pulses/s, or a frequency of 1000 Hz.

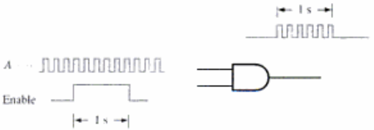


FIGURE 3–15

SECTION 3–2 REVIEW

1. When is the output of an AND gate HIGH?
2. When is the output of an AND gate LOW?
3. Describe the truth table for a 5-input AND gate.

3–3 THE OR GATE

The OR gate is another of the basic gates from which all logic functions are constructed. An OR gate can have two or more inputs and performs what is known as logical addition.

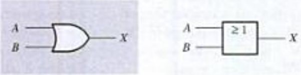
After completing this section, you should be able to

■ Identify an OR gate by its distinctive shape symbol or by its rectangular outline symbol ■ Describe the operation of an OR gate ■ Generate the truth table for an OR gate with any number of inputs ■ Produce a timing diagram for an OR gate with any specified input waveforms ■ Write the logic expression for an OR gate with any number of inputs ■ Discuss example of OR gate application

An **OR gate** has two or more inputs and one output, as indicated by the standard logic symbols in Figure 3–16, where OR gates with two inputs are illustrated. An OR gate can have any number of inputs greater than one. Although both distinctive shape and rectangular outline symbols are shown, the distinctive shape OR gate symbol is used in this textbook.

FIGURE 3–16

Standard logic symbol for OR gate



(a) Distinctive shape

(b) Rectangular outline with the OR (≥ 1) qualifying symbol

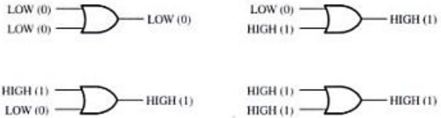
Operation of an OR Gate

An OR gate produces a HIGH on the output when *any* of the inputs is HIGH. The output is LOW only when all of the inputs are LOW. Therefore, an OR gate determines when one or more of its inputs are HIGH and produces a HIGH on its output to indicate this condition. The inputs of the 2-input OR gate in Figure 3–16 are labelled *A* and *B*, and the output is labelled *X*. The operation of the gate can be stated as follows:

For a 2-input OR gate, output *X* is HIGH if either input *A* or input *B* is HIGH, or if both *A* and *B* are HIGH; *X* is LOW if both *A* and *B* are LOW.

The HIGH level is the active or asserted output level for the OR gate. Figure 3–17 illustrates the operation for a 2-input OR gate for all four possible input combinations.

FIGURE 3–17



OR Gate Truth Table

The operation of a 2-input OR gate is described in Table 3–5. This truth table can be expanded for any number of inputs; but regardless of the number of inputs, the output is HIGH when one or more of the inputs are HIGH.

▶ TABLE 3–5

Truth table for OR gate

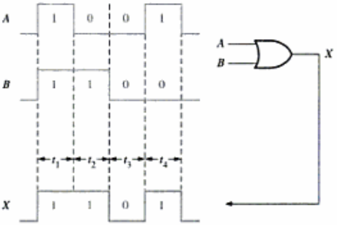
INPUTS		OUTPUT
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

1 = HIGH, 0 = LOW.

Pulsed Operation

Now, let us look at the operation of an OR gate with pulsed inputs, keeping in mind its logical operation. Again, the important thing in the analysis of gate operation with pulsed waveforms is the time relationship of all the waveforms involved. For example, in Figure 3–18, inputs *A* and *B* are both HIGH (1) during time interval t_1, making the output HIGH (1). During time interval t_2, input *A* is LOW (0), but because input *B* is HIGH (1), the output is HIGH (1). Both inputs are LOW (0) during time interval t_3, so there is a LOW (0) output during this time. During time interval t_4, the output is HIGH (1) because input *A* is HIGH (1).

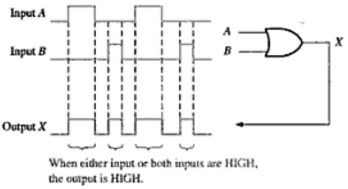
▶ FIGURE 3–18



In this illustration, we have simply applied the truth table operation of the OR gate to each of the time intervals during which the levels are nonchanging. Examples 3–6 through 3–8 further illustrate OR gate operation with waveforms on the inputs.

EXAMPLE 3–6

If the two input waveforms, *A* and *B*, in Figure 3–19 are applied to the OR gate, what is the resulting output waveform?


Input *A*
Input *B*
Output *X*
A
B
X
When either input or both inputs are HIGH, the output is HIGH.

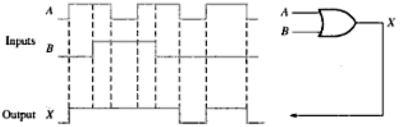
▲ FIGURE 3–19

Solution The output waveform *X* of a 2-input OR gate is HIGH when either or both input waveforms are HIGH as shown in the timing diagram. In this case, both input waveforms are never HIGH at the same time.

Supplementary Problem Determine the output waveform and show the timing diagram if input *A* is changed such that it is HIGH from the beginning of the existing first pulse to the end of the existing second pulse.

EXAMPLE 3–7

For the two input waveforms, *A* and *B*, in Figure 3–20, show the output waveform with its proper relation to the inputs.


A
Inputs
B
Output *X*
A
B
X

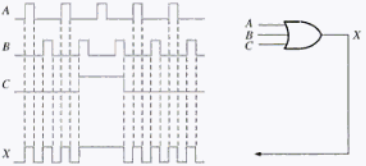
▲ FIGURE 3–20

Solution When either or both input waveforms are HIGH, the output is HIGH as shown by the output waveform *X* in the timing diagram.

Supplementary Problem Determine the output waveform and show the timing diagram if the middle pulse of input *A* is replaced by a LOW level.

EXAMPLE 3–8

For the 3-input OR gate in Figure 3–21, determine the output waveform in proper time relation to the inputs.



▲ FIGURE 3–21

Solution The output is HIGH when one or more of the input waveforms are HIGH as indicated by the output waveform X in the timing diagram.

Supplementary Problem Determine the output waveform and show the timing diagram if input C is always LOW.

Logic Expressions for an OR Gate

The logical OR function of two variables is represented mathematically by a + between the two variables, for example, $A + B$.

Addition in Boolean algebra involves variables whose values are either binary 1 or binary 0. The basic rules for Boolean addition are as follows:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

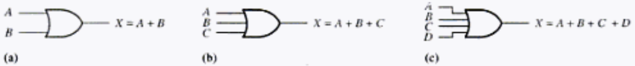
Boolean addition is the same as the OR function.

Notice that Boolean addition differs from binary addition in the case where two 1s are added. There is no carry in Boolean addition.

The operation of a 2-input OR gate can be expressed as follows: If one input variable is A, if the other input variable is B, and if the output variable is X, then the Boolean expression is

$$X = A + B$$

Figure 3–22(a) shows the gate logic symbol with input and output variables labelled.



(a) (b) (c)

▲ FIGURE 3–22

To extend the OR expression to more than two input variables, a new letter is used for each additional variable. For instance, the function of a 3-input OR gate can be expressed as $X = A + B + C$. The expression for a 4-input OR gate can be written as $X = A + B + C + D$, and so on. Parts (b) and (c) of Figure 3–22 show OR gates with three and four input variables, respectively.

OR gate operation can be evaluated by using the Boolean expressions for the output X by substituting all possible combinations of 1 and 0 values for the input variables, as shown in Table 3–6 for a 2-input OR gate. This evaluation shows that the output X of an OR gate is a 1 (HIGH) when any one or more of the inputs are 1 (HIGH). A similar analysis can be extended to OR gates with any number of input variables.

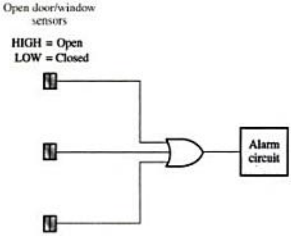
TABLE 3–6

A	B	$A + B = X$
0	0	$0 + 0 = 0$
0	1	$0 + 1 = 1$
1	0	$1 + 0 = 1$
1	1	$1 + 1 = 1$

Application Example

A simplified portion of an intrusion detection and alarm system is shown in Figure 3–23. This system could be used for one room in a home—a room with two windows and a door. The sensors are magnetic switches that produce a HIGH output when open and a LOW output when closed. As long as the windows and the door are secured, the switches are closed and all three of the OR gate inputs are LOW. When one of the windows or the door is opened, a HIGH is produced on that input to the OR gate and the gate output goes HIGH. It then activates and latches an alarm circuit to warn of the intrusion.

FIGURE 3–23



SECTION 3–3 REVIEW

1. When is the output of an OR gate HIGH?
2. When is the output of an OR gate LOW?
3. Describe the truth table for a 3-input OR gate.

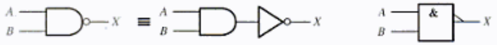
3–4 THE NAND GATE

The NAND gate is a popular logic element because it can be used as a universal gate; that is, NAND gates can be used in combination to perform the AND, OR, and inverter operations. The universal property of the NAND gate will be examined thoroughly in Chapter 5.

After completing this section, you should be able to

- Identify a NAND gate by its distinctive shape symbol or by its rectangular outline symbol ▪ Describe the operation of a NAND gate ▪ Develop the truth table for a NAND gate with any number of inputs ▪ Produce a timing diagram for a NAND gate with any specified input waveforms ▪ Write the logic expression for a NAND gate with any number of inputs ▪ Describe NAND gate operation in terms of its negative-OR equivalent ▪ Discuss examples of NAND gate applications

The term *NAND* is a contraction of NOT-AND and implies an AND function with a complemented (inverted) output. The standard logic symbol for a 2-input NAND gate and its equivalency to an AND gate followed by an inverter are shown in Figure 3–24(a), where the symbol ≡ means equivalent to. A rectangular outline symbol is shown in part (b).



(a) Distinctive shape, 2-input NAND gate and its NOT/AND equivalent

(b) Rectangular outline, 2-input NAND gate with polarity indicator

▲ **FIGURE 3–24**

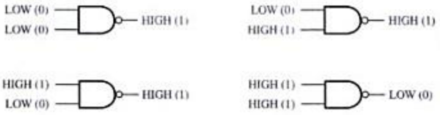
Standard NAND gate logic symbols

Operation of a NAND Gate

A NAND gate produces a LOW output only when all the inputs are HIGH. When any of the inputs is LOW, the output will be HIGH. For the specific case of a 2-input NAND gate, as shown in Figure 3–24 with the inputs labelled *A* and *B* and the output labelled *X*, the operation can be stated as follows:

> **For a 2-input NAND gate, output *X* is LOW if inputs *A* and *B* are HIGH; *X* is HIGH if either *A* or *B* is LOW, or if both *A* and *B* are LOW.**

Note that this operation is opposite that of the AND in terms of the output level. In a NAND gate, the LOW level (0) is the active or asserted output level, as indicated by the bubble on the output. Figure 3–25 illustrates the operation of a 2-input NAND gate for all four input combinations, and Table 3–7 is the truth table summarizing the logical operation of the 2-input NAND gate.


LOW (0)
LOW (0)
HIGH (1)
LOW (0)
HIGH (1)
HIGH (1)
HIGH (1)
LOW (0)
HIGH (1)
HIGH (1)
HIGH (1)
LOW (0)

▲ FIGURE 3-25

▶ TABLE 3-7

Truth table for NAND gate

INPUTS *A*	INPUTS *B*	OUTPUT *X*
0	0	1
0	1	1
1	0	1
1	1	0

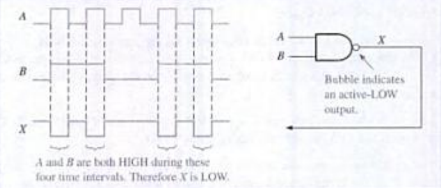
1 = HIGH, 0 = LOW

Pulsed Operation

Now, let us look at the pulsed operation of a NAND gate. Remember from the truth table that the only time a LOW output occurs is when all of the inputs are HIGH. Examples 3–9 and 3–10 illustrate pulsed operation.

EXAMPLE 3-9

If the two waveforms A and B shown in Figure 3–26 are applied to the NAND gate inputs, determine the resulting output waveform.


A
B
X
A
B
X
Bubble indicates an active-LOW output.
A and B are both HIGH during these four time intervals. Therefore X is LOW.

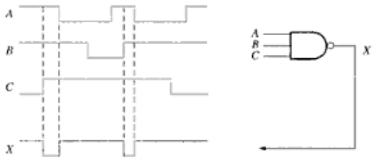
▲ FIGURE 3-26

Solution Output waveform X is LOW only during the four time intervals when both input waveforms A and B are HIGH as shown in the timing diagram.

Supplementary Problem Determine the output waveform and show the timing diagram if input waveform B is inverted.

EXAMPLE 3–10

Show the output waveform for the 3-input NAND gate in Figure 3–27 with its proper time relationship to the inputs.



▲ FIGURE 3–27

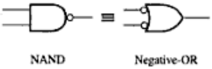
Solution The output waveform X is LOW only when all three input waveforms are HIGH as shown in the timing diagram.

Supplementary Problem Determine the output waveform and show the timing diagram if input waveform A is inverted.

***Negative-OR Equivalent Operation* of a NAND Gate** Inherent in a NAND gate's operation is the fact that one or more LOW inputs produce a HIGH output. Table 3–7 shows that output X is HIGH (1) when any of the inputs, A and B, are LOW (0). From this viewpoint, a NAND gate can be used for an OR operation that requires one or more LOW inputs to produce a HIGH output. This aspect of NAND operation is referred to as **negative-OR**. The term *negative* in this context means that the inputs are defined to be in the active or asserted state when LOW.

For a 2-input NAND gate performing a negative-OR operation, output X is HIGH if either input A or input B is LOW, or if both A and B are LOW.

When a NAND gate is used to detect one or more LOWs on its inputs rather than all HIGHs, it is performing the negative-OR operation and is represented by the standard logic symbol shown in Figure 3–28. Although the two symbols in Figure 3–28 represent the same physical gate, they serve to define its role or mode of operation in a particular application, as illustrated by Examples 3–11 through 3–13.



▲ FIGURE 3–28

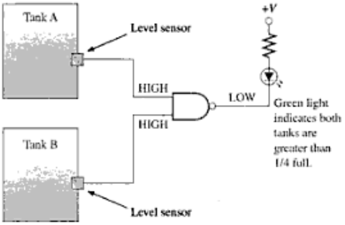
Standard symbols of NAND gate

EXAMPLE 3–11

A manufacturing plant uses two tanks to store a certain liquid chemical that is required in a manufacturing process. Each tank has a sensor that detects when the chemical level drops to 25% of full. The sensors produce a 5 V level when the tanks are more than one-quarter full. When the volume of chemical in a tank drops to one-quarter full, the sensor puts out a 0 V level.

It is required that a single green light-emitting diode (LED) on an indicator panel show when both tanks are more than one-quarter full. Show how a NAND gate can be used to implement this function.

Solution Figure 3–29 shows a NAND gate with its two inputs connected to the tank level sensors and its output connected to the indicator panel. The operation can be stated as follows: If tank *A and* tank *B* are above one-quarter full, the LED is on.



▲ **FIGURE 3–29**

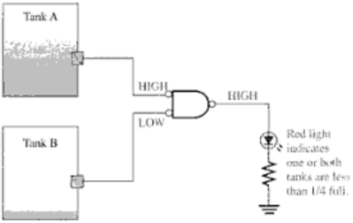
As long as *both* sensor outputs are HIGH (5 V), indicating that both tanks are more than one-quarter full, the NAND gate output is LOW (0 V). The green LED circuit is arranged so that a LOW voltage turns it on.

Supplementary Problem How can the circuit of Figure 3–29 be modified to monitor the levels in three tanks rather than two?

EXAMPLE 3–12

The supervisor of the manufacturing process described in Example 3–11 has decided that he would prefer to have a red LED display come on when at least one of the tanks fall to the quarter-full level rather than have the green LED display indicate when both are above one quarter. Show how this requirement can be implemented.

Solution Figure 3–30 shows a NAND gate operating as a negative-OR gate to detect the occurrence of at least one LOW on its inputs. A sensor puts out a LOW voltage if the volume in its tank goes to one-quarter full or less. When this happens, the gate output goes HIGH. The red LED circuit in the panel is arranged so that a HIGH voltage turns it on. The operation can be stated as follows: If tank *A or* tank *B or* both are below one-quarter full, the LED is on.


Tank A
HIGH
HIGH
LOW
Tank B
Red light indicates one or both tanks are less than 1/4 full.

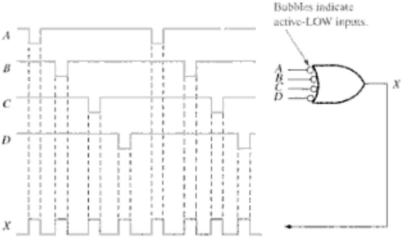
▲ FIGURE 3–30

Notice that, in this example and in Example 3–11, the same 2-input NAND gate is used, but a different gate symbol is used in the schematic, illustrating the different way in which the NAND and negative-OR operations are used.

Supplementary Problem How can the circuit in Figure 3–30 be modified to monitor four tanks rather than two?

EXAMPLE 3–13

For the 4-input NAND gate in Figure 3–31, operating as a negative-OR, determine the output with respect to the inputs.


Bubbles indicate active-LOW inputs.
A
B
C
D
X
A
B
C
D
X

▲ FIGURE 3–31

Solution The output waveform X is HIGH any time an input waveform is LOW as shown in the timing diagram.

Supplementary Problem Determine the output waveform if input waveform A is inverted before it is applied to the gate.

Logic Expressions for a NAND Gate

The Boolean expression for the output of a 2-input NAND gate is

$$X = \overline{AB}$$

This expression says that the two input variables, A and B, are first ANDed and then complemented, as indicated by the bar over the AND expression. This is a description in equation form of the operation of a NAND gate with two inputs. Evaluating this expression for all possible values of the two input variables, you get the results shown in Table 3–8.

▶ **TABLE 3–8**

A	B	$\overline{AB} = X$
0	0	$\overline{0 \cdot 0} = \overline{0} = 1$
0	1	$\overline{0 \cdot 1} = \overline{0} = 1$
1	0	$\overline{1 \cdot 0} = \overline{0} = 1$
1	1	$\overline{1 \cdot 1} = \overline{1} = 0$

Once an expression is determined for a given logic function, that function can be evaluated for all possible values of the variables. The evaluation tells you exactly what the output of the logic circuit is for each of the input conditions, and it therefore gives you a complete description of the circuit's logic operation. The NAND expression can be extended to more than two input variables by including additional letters to represent the other variables.

SECTION 3–4 REVIEW

1. When is the output of a NAND gate LOW?
2. When is the output of a NAND gate HIGH?
3. Describe the functional differences between a NAND gate and a negative-OR gate. Do they both have the same truth table?
4. Write the output expression for a NAND gate with inputs A, B and C.

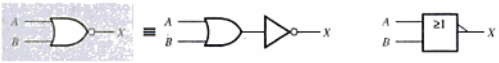
3–5 THE NOR GATE

The NOR gate, like the NAND gate, is a useful logic element because it can also be used as a universal gate; that is, NOR gates can be used in combination to perform the AND, OR, and inverter operations. The universal property of the NOR gate will be examined thoroughly in Chapter 5.

After completing this section, you should be able to

- Identify a NOR gate by its distinctive shape symbol or by its rectangular outline symbol
- Describe the operation of a NOR gate
- Develop the truth table for a NOR gate with any number of inputs
- Produce a timing diagram for a NOR gate with any specified input waveforms
- Write the logic expression for a NOR gate with any number of inputs
- Describe NOR gate operation in terms of its negative-AND equivalent
- Discuss examples of NOR gate applications

The term *NOR* is a contraction of NOT-OR and implies an OR function with an inverted (complemented) output. The standard logic symbol for a 2-input NOR gate and its equivalent OR gate followed by an inverter are shown in Figure 3–32(a). A rectangular outline symbol is shown in part (b).



(a) Distinctive shape, 2-input NOR gate and its NOT/OR equivalent

(b) Rectangular outline, 2-input NOR gate with polarity indicator

▲ **FIGURE 3–32**

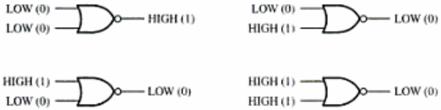
Standard NOR gate logic symbols.

Operation of a NOR Gate

A NOR gate produces a LOW output when *any* of its inputs is HIGH. Only when all of its inputs are LOW is the output HIGH. For the specific case of a 2-input NOR gate, as shown in Figure 3–32 with the inputs labelled *A* and *B* and the output labelled *X*, the operation can be stated as follows:

> **For a 2-input NOR gate, output *X* is LOW if either input *A* or input *B* is HIGH, or if both *A* and *B* are HIGH; *X* is HIGH if both *A* and *B* are LOW.**

This operation results in an output level opposite that of the OR gate. In a NOR gate, the LOW output is the active or asserted output level as indicated by the bubble on the output. Figure 3–33 illustrates the operation of a 2-input NOR gate for all four possible input combinations, and Table 3–9 is the truth table for a 2-input NOR gate.



▲ **FIGURE 3–33**

▶ **TABLE 3–9**

Truth table for NOR gate

INPUTS		OUTPUT
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0

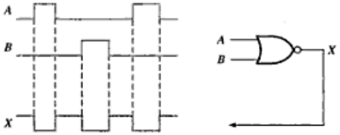
1 = HIGH, 0 = LOW.

Pulsed Operation

The next two examples illustrate the operation of a NOR gate with pulsed inputs. Again, as with the other types of gates, we will simply follow the truth table operation to determine the output waveforms in the proper time relationship to the inputs.

EXAMPLE 3–14

If the two waveforms shown in Figure 3–34 are applied to a NOR gate, what is the resulting output waveform?



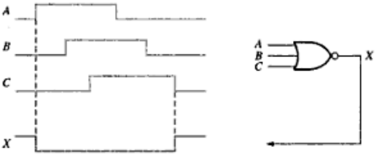
▲ **FIGURE 3–34**

Solution Whenever any input of the NOR gate is HIGH, the output is LOW as shown by the output waveform X in the timing diagram.

Supplementary Problem Invert input B and determine the output waveform in relation to the inputs.

EXAMPLE 3–15

Show the output waveform for the 3-input NOR gate in Figure 3–35 with the proper time relation to the inputs.



▲ **FIGURE 3–35**

Solution The output X is LOW when any input is HIGH as shown by the output waveform X in the timing diagram.

Supplementary Problem With the B and C inputs inverted, determine the output and show the timing diagram.

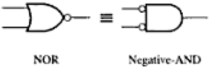
Negative-AND Equivalent Operation of the NOR Gate A NOR gate, like the NAND, has another aspect of its operation that is inherent in the way it logically functions. Table 3–9 shows that a HIGH is produced on the gate output only if all of the inputs are LOW. From this viewpoint, a NOR gate can be used for an AND operation that requires all LOW inputs to produce a HIGH output. This aspect of NOR operation is called

negative-AND. The term *negative* in this context means that the inputs are defined to be in the active or asserted state when LOW.

> **For a 2-input NOR gate performing a negative-AND operation, output *X* is HIGH if both inputs *A* and *B* are LOW.**

When a NOR gate is used to detect all LOWs on its inputs rather than one or more HIGHs, it is performing the negative-AND operation and is represented by the standard symbol in Figure 3–36. It is important to remember that the two symbols in Figure 3–36 represent the same physical gate and serve only to distinguish between the two modes of its operation. The following three examples illustrate this.

▶ **FIGURE 3–36**

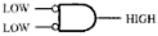


EXAMPLE 3–16

A device is needed to indicate when two LOW levels occur simultaneously on its inputs and to produce a HIGH output as an indication. Specify the device.

Solution A NOR gate operating as a negative-AND gate is required to produce a HIGH output when both inputs are LOW, as shown in Figure 3–37.

▶ **FIGURE 3–37**

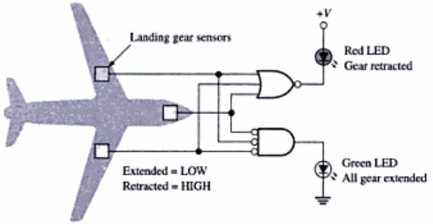


Supplementary Problem A device is needed to indicate when one or two HIGH levels occur on its inputs and to produce a LOW output as an indication. Specify the device.

EXAMPLE 3–17

As part of an aircraft's functional monitoring system, a circuit is required to indicate the status of the landing gears prior to landing. A green LED display turns on if all three gears are properly extended when the "gear down" switch has been activated in preparation for landing. A red LED display turns on if any of the gears fail to extend properly prior to landing. When a landing gear is extended, its sensor produces a LOW voltage. When a landing gear is retracted, its sensor produces a HIGH voltage. Implement a circuit to meet this requirement.

Solution Power is applied to the circuit only when the "gear down" switch its activated. Use a NOR gate for each of the two requirements as shown in Figure 3–38. One NOR gate operates as a negative-AND to detect a LOW from each of the three landing gear sensors. When all three of the gate inputs are LOW, the three landing gears are properly extended and the resulting HIGH output from the negative-AND gate turns on the green LED display. The other NOR gate operates as a NOR to detect if one or more of the landing gears remain retracted when the "gear down" switch is activated. When one or more of the landing gears remain retracted, the resulting HIGH from the sensor is detected by the NOR gate, which produces a LOW output to turn on the red LED warning display.

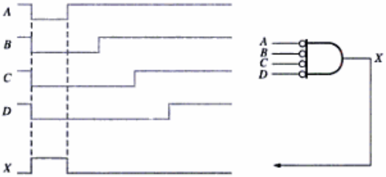

+V
Landing gear sensors
Red LED
Gear retracted
Green LED
All gear extended
Extended = LOW
Retracted = HIGH

▲ FIGURE 3–38

Supplementary Problem What type of gate should be used to detect if all three landing gears are retracted after takeoff, assuming a LOW output is required to activate an LED display?

EXAMPLE 3–18

For the 4-input NOR gate operating as a negative-AND in Figure 3–39, determine the output relative to the inputs.


A
B
C
D
X
A
B
C
D
X

▲ FIGURE 3–39

Solution Any time all of the input waveforms are LOW, the output is HIGH as shown by output waveform X in the timing diagram.

Supplementary Problem Determine the output with input D inverted and show the timing diagram.

Logic Expressions for a NOR Gate

The Boolean expression for the output of a 2-input NOR gate can be written as

$$X = \overline{A + B}$$

This equation says that the two input variables are first ORed and then complemented, as indicated by the bar over the OR expression. Evaluating this expression, you get the results shown in Table 3–10. The NOR expression can be extended to more than two input variables by including additional letters to represent the other variables.

TABLE 3–10

A	B	$\overline{A + B} = X$
0	0	$\overline{0 + 0} = \overline{0} = 1$
0	1	$\overline{0 + 1} = \overline{1} = 0$
1	0	$\overline{1 + 0} = \overline{1} = 0$
1	1	$\overline{1 + 1} = \overline{1} = 0$

SECTION 3–5 REVIEW

1. When is the output of a NOR gate HIGH?
2. When is the output of a NOR gate LOW?
3. Describe the functional difference between a NOR gate and a negative-AND gate. Do they both have the same truth table?
4. Write the output expression for a 3-input NOR with input variables *A*, *B*, and *C*.

3–6 THE EXCLUSIVE-OR AND EXCLUSIVE-NOR GATES

Exclusive-OR and exclusive-NOR gates are formed by a combination of other gates already discussed, as you will see in Chapter 5. However, because of their fundamental importance in many applications, these gates are often treated as basic logic elements with their own unique symbols.

After completing this section, you should be able to

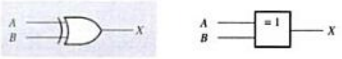
- Identify the exclusive-OR and exclusive-NOR gates by their distinctive shape symbols or by their rectangular outline symbols
- Describe the operations of exclusive-OR and exclusive-NOR gates
- Show the truth tables for exclusive-OR and exclusive-NOR gates
- Produce a timing diagram for an exclusive-OR or exclusive-NOR gate with any specified input waveforms
- Discuss examples of exclusive-OR and exclusive-NOR gate applications

The Exclusive-OR Gate

Standard symbols for an exclusive-OR (XOR for short) gate are shown in Figure 3–40. The XOR gate has only two inputs.

FIGURE 3–40

Standard logic symbols for the exclusive-OR gate



(a) Distinctive shape

(b) Rectangular outline with the XOR

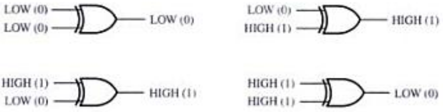
The output of an **exclusive-OR gate** is HIGH *only* when the two inputs are at opposite logic levels. This operation can be stated as follows with reference to inputs *A* and *B* and output *X*:

For an exclusive-OR gate, output *X* is HIGH if input *A* is LOW and input *B* is HIGH, or if input *A* is HIGH and input *B* is LOW; *X* is LOW if *A* and *B* are both HIGH or both LOW.

The four possible input combinations and the resulting outputs for an XOR gate are illustrated in Figure 3–41. The HIGH level is the active or asserted output level and occurs only when the inputs are at opposite levels. The operation of an XOR gate is summarized in the truth table shown in Table 3–11.

▶ **FIGURE 3–41**

All possible logic levels for an exclusive-OR gate



▶ **TABLE 3–11**

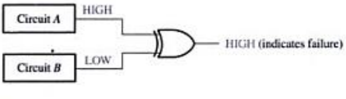
Truth table for an exclusive-OR gate

INPUTS		OUTPUT
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	0

EXAMPLE 3–19

A certain system contains two identical circuits operating in parallel. As long as both are operating properly, the outputs of both circuits are always the same. If one of the circuits fails, the outputs will be at opposite levels at some time. Devise a way to detect that a failure has occurred in one of the circuits.

Solution The outputs of the circuits are connected to the inputs of an XOR gate as shown in Figure 3–42. A failure in either one of the circuits produces differing outputs, which cause the XOR inputs to be at opposite levels. This condition produces a HIGH on the output of the XOR gate, indicating a failure in one of the circuits.



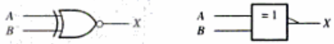
▲ **FIGURE 3–42**

Supplementary Problem Will the exclusive-OR gate always detect simultaneous failures in both circuits of Figure 3–42? If not, under what condition?

The Exclusive-NOR Gate

Standard symbols for an **exclusive-NOR** (XNOR) **gate** are shown in Figure 3–43. Like the XOR gate, an XNOR has only two inputs. The bubble on the output of the XNOR symbol indicates that its output is opposite that of the XOR gate. When the two input logic levels are opposite, the output of the exclusive-NOR gate is LOW. The operation can be stated as follows (*A* and *B* are inputs, *X* is output):

> **For an exclusive-NOR gate, output *X* is LOW if input *A* is LOW and input *B* is HIGH, or if *A* is HIGH and *B* is LOW; *X* is HIGH if *A* and *B* are both HIGH or both LOW.**

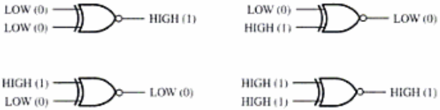


(a) Distinctive shape (b) Rectangular outline

▲ **FIGURE 3–43**

Standard logic symbols for the exclusive-NOR gate

The four possible input combinations and the resulting outputs for an XNOR gate are shown in Figure 3–44. The operation of an XNOR gate is summarized in Table 3–12. Notice that the output is HIGH when the same level is on both inputs.



▲ **FIGURE 3–44**

► **TABLE 3–12**

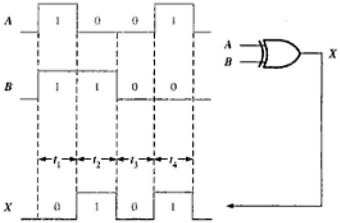
Truth table for an exclusive-NOR gate

INPUTS		OUTPUT
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1

Pulsed Operation

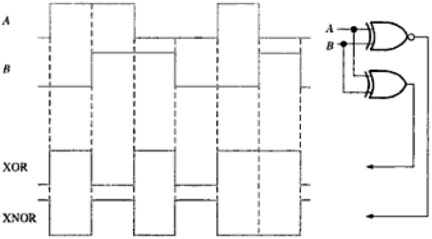
As we have done with the other gates, let us examine the operation of XOR and XNOR gates under pulsed input conditions. As before, we apply the truth table operation during each distinct time interval of the pulsed inputs, as illustrated in Figure 3–45 for an XOR gate. You can see that the input waveforms *A* and *B* are at opposite levels during time intervals t_2 and t_4. Therefore, the output *X* is HIGH during these two times. Since both inputs are at the same level, either both HIGH or both LOW, during time intervals t_1 and t_3, the output is LOW during those times as shown in the timing diagram.

◄ **FIGURE 3–45**



EXAMPLE 3–20

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, *A* and *B*, in Figure 3–46.



▲ **FIGURE 3–46**

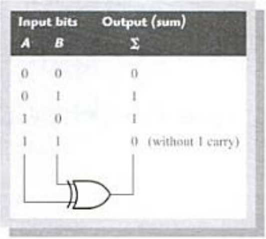
Solution The output waveforms are shown in Figure 3–46. Notice that the XOR output is HIGH only when both inputs are at opposite levels. Notice that the XNOR output is HIGH only when both inputs are the same.

Supplementary Problem Determine the output waveforms if the two input waveforms, *A* and *B*, are inverted.

Application Example

An exclusive-OR gate can be used as a two-bit adder. Recall from Chapter 2 that the basic rules for binary addition are as follows: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 10$. An examination of the truth table for an XOR gate will show you that its output is the binary sum of the two input bits. In the case where the inputs are both 1s, the output is the sum 0, but you lose the carry of 1. In Chapter 6 you will see how XOR gates are combined to make complete adding circuits. Figure 3–47 illustrates an XOR gate used as a basic adder.

▶ FIGURE 3–47



Input bits		Output (sum)
A	*B*	Σ
0	0	0
0	1	1
1	0	1
1	1	0 (without 1 carry)

SECTION 3–6 REVIEW

1. When is the output of an XOR gate HIGH?
2. When is the output of an XNOR gate HIGH?
3. How can you use an XOR gate to detect when two bits are different?

3–7 EXAMPLES OF IC GATES

There are three digital integrated circuit (IC) technologies that are used to implement the basic logic gates. Two of these, CMOS and TTL, are the most widely used and the third, ECL, is used in more specialized applications. The logic operations of NOT, AND, OR, NAND, NOR, and exclusive-OR are the same regardless of the IC technology used; that is, an AND gate has the same logic function whether it is implemented with CMOS, TTL, or ECL. We will not cover the IC technologies in this section.

After completing this section, you should be able to

- Understand available ICs for implementation of gate functions
- Understand pin configuration diagrams and logic symbols

CMOS stands for Complementary Metal-Oxide Semiconductor and is implemented with a type of field-effect transistor. **TTL** stands for Transistor-Transistor Logic and is implemented with bipolar junction transistors. **ECL**, Emitter-Coupled Logic, is also a **bipolar** technology. The coverage in this section is restricted to only the commercially available gate ICs, their pin configuration diagrams, and logic symbols. The IC technologies at the circuit component level and their electrical characteristics will be covered in Chapter 11.

Available ICs for Gates

All the logic functions introduced in this chapter are commercially available in integrated circuit (IC) form. Table 3–13 gives some of the available gate ICs.

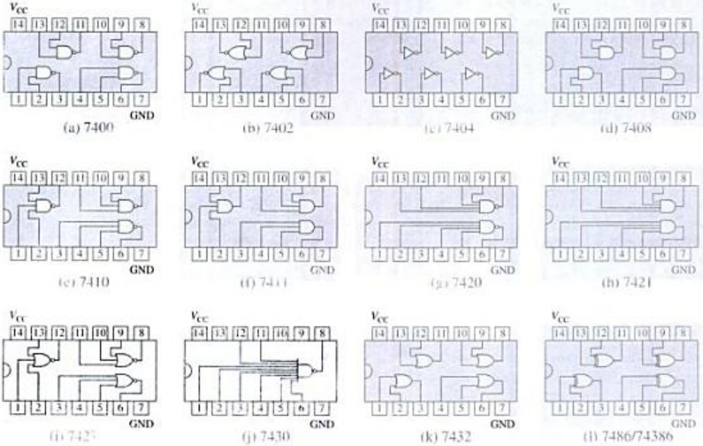
Pin Configuration Diagrams

All the basic logic operations, NOT, AND, OR, NAND, NOR, and Exclusive-OR (XOR) in IC form are available in DIP. The pin configuration diagrams of most of the devices given in Table 3–13 are shown in Figure 3–48. From the pin configuration diagrams, we observe that

these ICs contain multiple identical gates. For **example, 7400 IC chip** is a quadruple 2-input **NAND gate** available in 14-pin DIP. It has four **identical, independent 2-input NAND gates arranged as** shown in Figure 3–48(a). It requires **supply voltage to be connected between** V_{CC} **and GND** pins for the proper operation of gates.

TABLE 3-13

IC NO.	DESCRIPTION
7400	Quad 2-input NAND gates
7402	Quad 2-input NOR gates
7404	Hex inverters
7408	Quad 2-input AND gates
7410	Triple 3-input NAND gates
7411	Triple 3-input AND gates
7420	Dual 4-input NAND gates
7421	Dual 4-input AND gates
7427	Triple 3-input NOR gates
7430	8-input NAND gate
7432	Quad 2-input OR gates
7486, 74386	Quad EX-OR gates
74133	13-input NAND gate
74135	Quad EX-OR/NOR gates
74260	Dual 5-input NOR gates



(a) 7400 (b) 7402 (c) 7404 (d) 7408

(e) 7410 (f) 7411 (g) 7420 (h) 7421

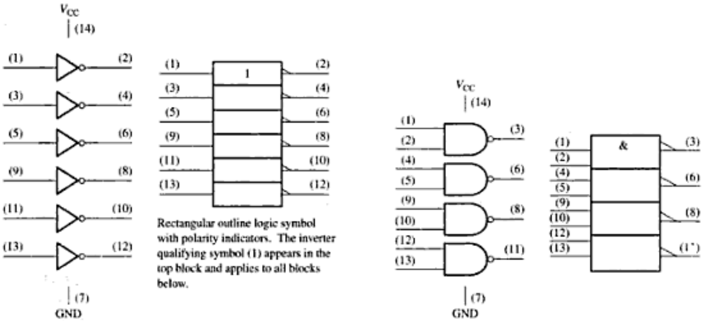
(i) 7427 (j) 7430 (k) 7432 (l) 7486/74386

FIGURE 3-48

Pin configuration diagrams of some gate ICs

Logic Symbols

The logic symbols for the gate ICs use the standard gate symbols and show the number of gates in the IC package and the associated pin numbers for each gate as well as the pin numbers for V_{CC} and ground. An example is shown in Figure 3–49 for a hex inverter and for a quad 2-input NAND gate. Both the distinctive shape and the rectangular outline formats are shown. Regardless of the logic family, all devices with the same suffix are pin-compatible; in other words, they will have the same arrangement of pin numbers. For example, the 7400, 74S00, 74LS00, 74ALS00, 74F00, 74HC00, and 74AHC00 are all pin-compatible quad 2-input NAND gate packages.



Rectangular outline logic symbol with polarity indicators. The inverter qualifying symbol (1) appears in the top block and applies to all blocks below.

Distinctive shape logic diagram

(a) Hex inverter

(b) Quad 2-input NAND

▲ **FIGURE 3–49**

Logic symbols for hex inverter and quad 2-input NAND gate ICs

SECTION 3–7 REVIEW

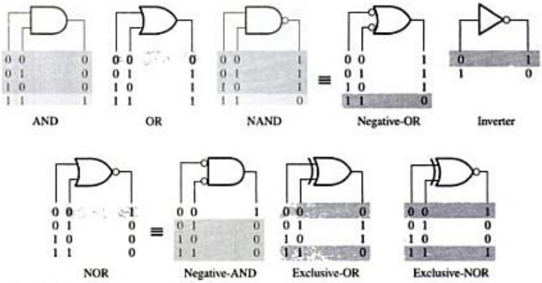
1. What is the number of 3-input NAND gates in a 14-pin IC?
2. What is the number of pins required in an IC package for four 2-input OR gates?
3. What will be the number of pins required in an IC package for a 13-input NAND gate?

SUMMARY

- The inverter output is the complement of the input.
- The AND gate output is HIGH only if all the inputs are HIGH.
- The OR gate output is HIGH if any of the inputs is HIGH.
- The NAND gate output is LOW only if all the inputs are HIGH.
- The NAND can be viewed as a negative-OR whose output is HIGH when any input is LOW.
- The NOR gate output is LOW if any of the inputs is HIGH.

- The NOR can be viewed as a negative-AND whose output is HIGH only if all the inputs are LOW.
- The exclusive-OR gate output is HIGH when the inputs are not the same.
- The exclusive-NOR gate output is LOW when the inputs are not the same.
- Distinctive shape symbols and truth tables for various logic gates (limited to 2 inputs) are shown in Figure 3–50.

FIGURE 3–50



Note: Active states are shown as unshaded.

- For basic logic functions, integrated circuits are commercially available in DIP.

SELF-TEST

Answers are at the end of the chapter.

1. When the input to an inverter is HIGH (1), the output is
   (a) HIGH or 1 **(b)** LOW or 1 **(c)** HIGH or 0 **(d)** LOW or 0
2. An inverter performs an operation known as
   (a) complementation **(b)** assertion
   (c) inversion **(d)** both answers (a) and (c)
3. The output of an AND gate with inputs A, B, and C is a 1 (HIGH) when
   (a) $A = 1, B = 1, C = 1$ **(b)** $A = 1, B = 0, C = 1$ **(c)** $A = 0, B = 0, C = 0$
4. The output of an OR gate with inputs A, B, and C is a 1 (HIGH) when
   (a) $A = 1, B = 1, C = 1$ **(b)** $A = 0, B = 0, C = 1$ **(c)** $A = 0, B = 0, C = 0$
   (d) answers (a), (b), and (c) **(e)** only answers (a) and (b)
5. A pulse is applied to each input of a 2-input NAND gate. One pulse goes HIGH at $t = 0$ and goes back LOW at $t = 1$ ms. The other pulse goes HIGH at $t = 0.8$ ms and goes back LOW at $t = 3$ ms. The output pulse can be described as follows:
   (a) It goes LOW at $t = 0$ and back HIGH at $t = 3$ ms.
   (b) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 3$ ms.
   (c) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 1$ ms.
   (d) It goes LOW at $t = 0.8$ ms and back LOW at $t = 1$ ms.

6. A pulse is applied to each input of a 2-input NOR gate. One pulse goes HIGH at $t = 0$ and goes back LOW at $t = 1$ ms. The other pulse goes HIGH at $t = 0.8$ ms and goes back LOW at $t = 3$ ms. The output pulse can be described as follows:

(a) It goes LOW at $t = 0$ and back HIGH at $t = 3$ ms.
(b) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 3$ ms.
(c) It goes LOW at $t = 0.8$ ms and back HIGH at $t = 1$ ms.
(d) It goes HIGH at $t = 0.8$ ms and back LOW at $t = 1$ ms.

7. A pulse is applied to each input of an exclusive-OR gate. One pulse goes HIGH at $t = 0$ and goes back LOW at $t = 1$ ms. The other pulse goes HIGH at $t = 0.8$ ms and goes back LOW at $t = 3$ ms. The output pulse can be described as follows:

(a) It goes HIGH at $t = 0$ and back LOW at $t = 3$ ms.
(b) It goes HIGH at $t = 0$ and back LOW at $t = 0.8$ ms.
(c) It goes HIGH at $t = 1$ ms and back LOW at $t = 3$ ms.
(d) both answers (b) and (c)

8. For and AND gate

(a) All LOW inputs produce a HIGH output
(b) Output is HIGH if and only if all inputs are HIGH
(c) Output is LOW if and only if all inputs are HIGH
(d) Output is LOW if and only if all inputs are LOW

9. The output of a gate is LOW when atleast one of its inputs is HIGH. This is true for

(a) AND **(b)** NAND **(c)** OR **(d)** NOR

10. The output of a gate is LOW when atleast one of its inputs is LOW. It is true for

(a) AND **(b)** OR **(c)** NAND **(d)** NOR

11. The output of a gate is HIGH when atleast one of its inputs is LOW. It is true for

(a) XOR **(b)** NAND **(c)** NOR **(d)** OR

12. The output of a gate is HIGH when atleast one of its inputs is HIGH. It is true for

(a) NAND **(b)** AND **(c)** OR **(d)** XOR

13. The output of a gate is HIGH if and only if all its inputs are HIGH. It is true for

(a) XOR **(b)** AND **(c)** OR **(d)** NAND

14. The output of a gate is LOW if and only if all its inputs are HIGH. It is true for

(a) AND **(b)** XNOR **(c)** NOR **(d)** NAND

15. The output of a gate is HIGH if and only if all its inputs are LOW. It is true for

(a) NOR **(b)** XOR **(c)** NAND **(d)** XNOR

16. The output of a gate is LOW if and only if all its inputs are LOW. It is true for

(a) XOR **(b)** AND **(c)** OR **(d)** NOR

17. The output of a 2-input gate is 1 if and only if its inputs are unequal.. It is true for

(a) OR **(b)** XOR **(c)** XNOR **(d)** NOR

18. The output of a 2-input gate is 0 if and only if its inputs are unequal. It is true for

(a) XNOR **(b)** AND **(c)** NOR **(d)** NAND

19. The output of a 2-input gate is 1 if and only if its inputs are equal. It is true for

(a) AND **(b)** XOR **(c)** OR **(d)** XNOR

20. The output of a 2-input gate is 0 if and only if its inputs are equal. It is true for

(a) AND **(b)** XOR **(c)** OR **(d)** NOR

21. The most suitable gate for comparing two bits is

(a) AND **(b)** OR **(c)** NAND **(d)** XOR

22. Which of the following gates can be used as an inverter?

(a) AND **(b)** OR **(c)** XOR **(d)** None of the above

23. Which of the following gates can not be used as an inverter?

(a) NAND (b) AND (c) NOR (d) XNOR

24. The maximum number of 3-input gates in a 16-pin IC will be

(a) 2 (b) 3 (c) 4 (d) 5

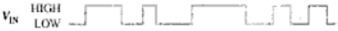
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 3-1 The Inverter

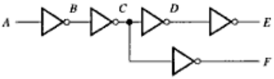
1. The input waveform shown in Figure 3–51 is applied to an inverter. Draw the timing diagram of the output waveform in proper relation to the input.

FIGURE 3-51



2. A network of cascaded inverters is shown in Figure 3–52. If a HIGH is applied to point *A*, determine the logic levels at points *B* through *F*.

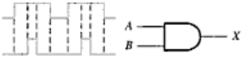
FIGURE 3-52



SECTION 3-2 The AND Gate

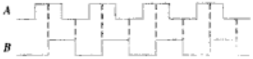
3. Determine the output, *X*, for a 2-input AND gate with the input waveforms shown in Figure 3–53. Show the proper relationship of output to inputs with a timing diagram.

FIGURE 3-53



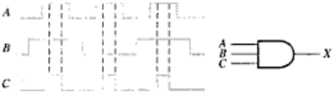
4. Repeat Problem 3 for the waveforms in Figure 3–54.

FIGURE 3-54



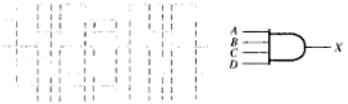
5. The input waveforms applied to a 3-input AND gate are as indicated in Figure 3–55. Show the output waveform in proper relation to the inputs with a timing diagram.

FIGURE 3-55



6. The input waveforms applied to a 4-input AND gate are as indicated in Figure 3–56. Show the output waveform in proper relation to the inputs with a timing diagram.

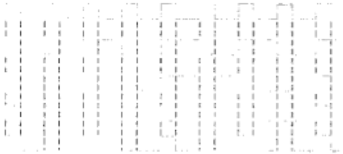
FIGURE 3–56



The OR Gate

7. Determine the output for a 2-input OR gate when the input waveforms are as in Figure 3–54 and draw a timing diagram.
8. Repeat Problem 5 for a 3-input OR gate.
9. Repeat Problem 6 for a 4-input OR gate.
10. For the five input waveforms in Figure 3–57, determine the output for a 5-input AND gate and the output for a 5-input OR gate. Draw the timing diagram.

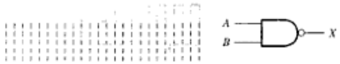
FIGURE 3–57



The NAND Gate

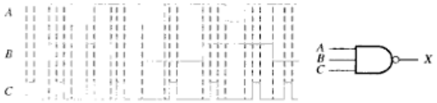
11. For the set of input waveforms in Figure 3–58, determine the output for the gate shown and draw the timing diagram.

FIGURE 3–58



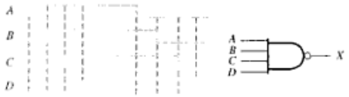
12. Determine the gate output for the input waveforms in Figure 3–59 and draw the timing diagram.

FIGURE 3–59



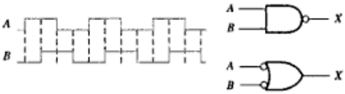
13. Determine the output waveform in Figure 3–60.

FIGURE 3–60



14. As you have learned, the two logic symbols shown in Figure 3–61 represent equivalent operations. The difference between the two is strictly from a functional viewpoint. For the NAND symbol, look for two HIGHs on the inputs to give a LOW output. For the negative-OR, look for at least one LOW on the inputs to give a HIGH on the output. Using these two functional viewpoints, show that each gate will produce the same output for the given inputs.

▶ **FIGURE 3–61**

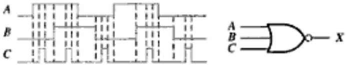


SECTION 3–5 The NOR Gate

15. Repeat Problem 11 for a 2-input NOR gate.

16. Determine the output waveform in Figure 3–62 and draw the timing diagram.

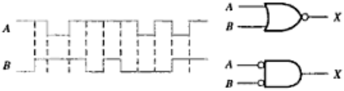
▶ **FIGURE 3–62**



17. Repeat Problem 13 for a 4-input NOR gate.

18. The NAND and the negative-OR symbols represent equivalent operations, but they are functionally different. For the NOR symbol, look for at least one HIGH on the inputs to give a LOW on the output. For the negative-AND, look for two LOWs on the inputs to give a HIGH output. Using these two functional points of view, show that both gates in Figure 3–63 will produce the same output for the given inputs.

▶ **FIGURE 3–63**



SECTION 3–6 The Exclusive-OR and Exclusive-NOR Gates

19. How does an exclusive-OR gate differ from an OR gate in its logical operation?

20. Repeat Problem 11 for an exclusive-OR gate.

21. Repeat Problem 11 for an exclusive-NOR gate.

22. Determine the output of an exclusive-OR gate for the inputs shown in Figure 3–54 and draw a timing diagram.

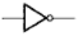
ANSWERS

SECTION REVIEWS

SECTION 3–1 The Inverter

1. When the inverter input is 1, the output is 0.

2. (a)



(b) A negative-going pulse is on the output (HIGH to LOW and back HIGH).

SECTION 3–2 The AND Gate

1. An AND gate output is HIGH when all inputs are HIGH.
2. An AND gate output is LOW when one or more inputs are LOW.
3. Five-input AND: $X = 1$ when $ABCDE = 11111$, and $X = 0$ for all other combinations of $ABCDE$.

SECTION 3–3 The OR Gate

1. An OR gate output is HIGH when one or more inputs are HIGH.
2. An OR gate output is LOW when all inputs are LOW.
3. Three-input OR: $X = 0$ when $ABC = 000$, and $X = 1$ for all other combinations of ABC.

SECTION 3–4 The NAND Gate

1. A NAND output is LOW when all inputs are HIGH.
2. A NAND output is HIGH when one or more inputs are LOW.
3. NAND: active-LOW output for all HIGH inputs; negative-OR: active-HIGH output for one or more LOW inputs. They have the same truth tables.
4. $X = \overline{ABC}$

SECTION 3–5 The NOR Gate

1. A NOR output is HIGH when all inputs are LOW.
2. A NOR output is LOW when one or more inputs are HIGH.
3. NOR: active-LOW output for one or more HIGH inputs; negative-AND: active-HIGH output for all LOW inputs. They have the same truth tables.
4. $X = \overline{A + B + C}$

SECTION 3–6 The Exclusive-OR Gate and Exclusive-NOR Gates

1. An XOR output is HIGH when the inputs are at opposite levels.
2. An XNOR output is HIGH when the inputs are at the same levels.
3. Apply the bits to the XOR inputs; when the output is HIGH, the bits are different.

SECTION 3–7 Examples of IC Gates

1. 3
2. 14
3. 16

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

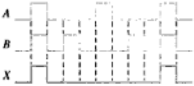
3–1 The timing diagram is not affected. **3–2** See Table 3–14.

▶ TABLE 3–14

INPUTS *ABCD*	OUTPUT *X*	INPUTS *ABCD*	OUTPUT *X*
0000	0	1000	0
0001	0	1001	0
0010	0	1010	0
0011	0	1011	0
0100	0	1100	0
0101	0	1101	0
0110	0	1110	0
0111	0	1111	1

3–3 See Figure 3–64.

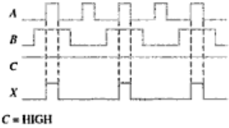
▶ FIGURE 3–64



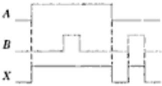
3–4 The output waveform is the same as input *A*.

3–5 See Figure 3–65.

3–6 See Figure 3–66.



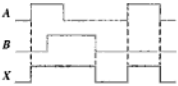
▲ FIGURE 3–65



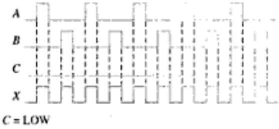
▲ FIGURE 3–66

3–7 See Figure 3–67.

3–8 See Figure 3–68.



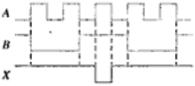
▲ FIGURE 3–67



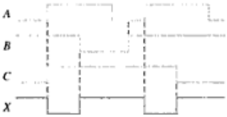
▲ FIGURE 3–68

3–9 See Figure 3–69.

3–10 See Figure 3–70.



▲ FIGURE 3–69



▲ FIGURE 3–70

3–11 Use a 3-input NAND gate.

3–12 Use a 4-input NAND gate operating as a negative-OR gate.

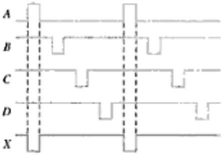
3–13 See Figure 3–71.

3–14 See Figure 3–72.

3–15 See Figure 3–73.

3–16 Use a 2-input NOR gate.

FIGURE 3-71



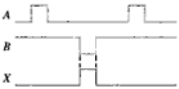


FIGURE 3-72

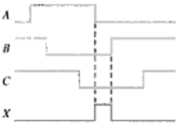


FIGURE 3-73

3–17 A 3-input NAND gate.

3–18 The output is always LOW. The timing diagram is a straight line.

3–19 The exclusive-OR gate will not detect simultaneous failures if both circuits produce the same outputs.

3–20 The outputs are unaffected.

SELF-TEST

1. (d) **2.** (d) **3.** (a) **4.** (e) **5.** (c) **6.** (a) **7.** (d) **8.** (b)
9. (d) **10.** (a) **11.** (b) **12.** (c) **13.** (b) **14.** (d) **15.** (a) **16.** (c)
17. (b) **18.** (a) **19.** (d) **20.** (b) **21.** (d) **22.** (c) **23.** (b) **24.** (b)

4

BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

CHAPTER OBJECTIVES

- Apply the basic laws and rules of Boolean algebra
- Apply DeMorgan's theorems to Boolean expressions
- Describe gate networks with Boolean expressions
- Evaluate Boolean expressions
- Simplify expressions by using the laws and rules of Boolean algebra
- Convert any Boolean expression into a sum-of-products (SOP) form
- Convert any Boolean expression into a product of-sums (POS) form
- Use a Karnaugh map to simplify Boolean expressions
- Use a Karnaugh map to simplify truth table functions
- Utilize "don't care" conditions to simplify logic functions
- Apply Boolean algebra and the Karnaugh map method to a system application

INTRODUCTION

In 1854, George Boole published a work titled *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities.* It was in this publication that a "logical algebra," known today as Boolean algebra, was formulated. Boolean algebra is a convenient and systematic way of expressing and analyzing the operation of logic circuits. Claude Shannon was the first to apply Boole's work to the analysis and design of logic circuits. In 1938, Shannon wrote a thesis at MIT titled *A Symbolic Analysis of Relay and Switching Circuits.*

This chapter covers the laws, rules, and theorems of Boolean algebra and their application to digital circuits. You will learn how to define a given circuit with a Boolean expression and then evaluate its operation. You will also learn how to simplify logic circuits using the methods of Boolean algebra and Karnaugh maps.

4-1 BOOLEAN OPERATIONS AND EXPRESSIONS

Boolean algebra is the mathematics of digital systems. A basic knowledge of Boolean algebra is indispensable to the study and analysis of logic circuits. In the last chapter, Boolean operations and expressions in terms of their relationship to NOT, AND, OR, NAND, and NOR gates were introduced. This section reviews that material and provides additional definitions and information.

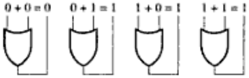
After completing this section, you should be able to

- Define *variable*
- Define *literal*
- Identify a sum term
- Evaluate a sum term
- Identify a product term
- Evaluate a product term
- Explain Boolean addition
- Explain Boolean multiplication

Variable, *complement*, and *literal* are terms used in Boolean algebra. A **variable** is a symbol (usually an italic uppercase letter) used to represent a logical quantity. Any single variable can have a 1 or a 0 value. The **complement** is the inverse of a variable and is indicated by a bar over the variable (overbar). For example, the complement of the variable A is $\overline{A}$. If $A = 1$, then $\overline{A} = 0$. If $A = 0$, then $\overline{A} = 1$. The complement of the variable A is read as "not A" or "A bar." Sometimes a prime symbol rather than an overbar is used to denote the complement of a variable; for example, B' indicates the complement of B. In this book, only the overbar is used. A **literal** is a variable or the complement of a variable.

Boolean Addition

Recall from Chapter 3 that **Boolean addition** is equivalent to the OR operation and the basic rules are illustrated with their relation to the OR gate as follows:



In Boolean algebra, a **sum term** is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are $A + B$, $A + \overline{B}$, $A + B + \overline{C}$, and $\overline{A} + B + C + \overline{D}$.

A sum term is equal to 1 when one or more of the literals in the term are 1. A sum term is equal to 0 only if each of the literals is 0.

EXAMPLE 4–1

Determine the values of A, B, C, and D that make the sum term $A + \overline{B} + C + \overline{D}$ equal to 0.

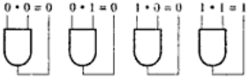
Solution For the sum term to be 0, each of the literals in the term must be 0. Therefore, $A = \mathbf{0}$, $B = \mathbf{1}$ so that $\overline{B} = 0$, $C = \mathbf{0}$, and $D = \mathbf{1}$ so that $\overline{D} = 0$.

$$A + \overline{B} + C + \overline{D} = 0 + \overline{1} + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

Supplementary Problem Determine the values of A and B that make the sum term $\overline{A} + B$ equal to 0.

Boolean Multiplication

Also recall from Chapter 3 that **Boolean multiplication** is equivalent to the AND operation and the basic rules are illustrated with their relation to the AND gate as follows:



In Boolean algebra, a **product term** is the product of literals. In logic circuits, a product term is produced by an AND operation with no OR operations involved. Some examples of product terms are AB, $A\overline{B}$, ABC, and $A\overline{B}C\overline{D}$.

A product term is equal to 1 only if each of the literals in the term is 1. A product term is equal to 0 when one or more of the literals are 0.

EXAMPLE 4–2

Determine the values of A, B, C, and D that make the product term $A\overline{B}C\overline{D}$ equal to 1.

Solution For the product term to be 1, each of the literals in the term must be 1. Therefore, $A = \mathbf{1}$, $B = \mathbf{0}$ so that $\overline{B} = 1$, $C = \mathbf{1}$, and $D = \mathbf{0}$ so that $\overline{D} = 1$.

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

Supplementary Problem Determine the values of A and B that make the product term $\overline{A}\,\overline{B}$ equal to 1.

SECTION 4–1 REVIEW

Answers are at the end of the chapter.

1. If $A = 0$, what does $\overline{A}$ equal?
2. Determine the values of A, B, and C that make the sum term $\overline{A} + \overline{B} + C$ equal to 0.
3. Determine the values of A, B, and C that make the product term $A\overline{B}C$ equal to 1.

4–2 LAWS AND RULES OF BOOLEAN ALGEBRA

As in other areas of mathematics, there are certain well-developed rules and laws that must be followed in order to properly apply Boolean algebra. The most important of these are presented in this section.

After completing this section, you should be able to

- Apply the commutative laws of addition and multiplication
- Apply the associative laws of addition and multiplication
- Apply the distributive law
- Apply twelve basic rules of Boolean algebra

Laws of Boolean Algebra

The basic laws of Boolean algebra—the **commutative laws** for addition and multiplication, the **associative laws** for addition and multiplication, and the **distributive law**—are the same as in ordinary algebra. Each of the laws is illustrated with two or three variables, but the number of variables is not limited to this.

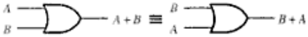
Commutative Laws The *commutative law of addition* for two variables is written as

Equation 4–1

$$A + B = B + A$$

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. Figure 4–1 illustrates the commutative law as applied to the OR gate and shows that it does not matter to which input each variable is applied. (The symbol ≡ means "equivalent to.")

FIGURE 4–1

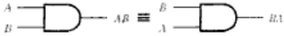


The *commutative law of multiplication* for two variables is

$AB = BA$ **Equation 4–2**

This law states that the order in which the variables are ANDed makes no difference. Figure 4–2 illustrates this law as applied to the AND gate.

FIGURE 4–2

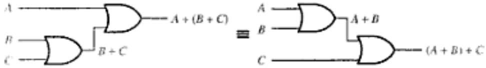


Associative Laws The *associative law of addition* is written as follows for three variables:

$A + (B + C) = (A + B) + C$ **Equation 4–3**

This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Figure 4–3 illustrates this law as applied to 2-input OR gates.

FIGURE 4–3

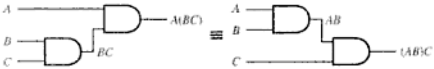


The *associative law of multiplication* is written as follows for three variables:

$A(BC) = (AB)C$ **Equation 4–4**

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Figure 4–4 illustrates this law as applied to 2-input AND gates.

FIGURE 4–4

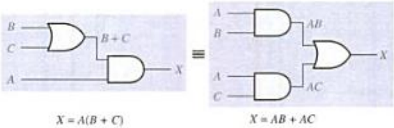


Distributive Law The distributive law is written for three variables as follows:

$A(B + C) = AB + AC$ **Equation 4–5**

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of *factoring* in which the common variable A is factored out of the product terms, for example, $AB + AC = A(B + C)$. Figure 4–5 illustrates the distributive law in terms of gate implementation.

▶ **FIGURE 4-5**



Rules of Boolean Algebra

Table 4–1 lists 12 basic rules that are useful in manipulating and simplifying **Boolean expressions**. Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.

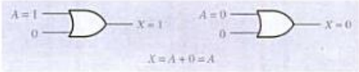
▶ **TABLE 4-1**

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = A$
5. $A + A = A$
6. $A + \overline{A} = 1$
7. $A \cdot A = A$
8. $A \cdot \overline{A} = 0$
9. $\overline{\overline{A}} = A$
10. $A + AB = A$
11. $A + \overline{A}B = A + B$
12. $(A + B)(A + C) = A + BC$

A, *B*, or *C* can represent a single variable or a combination of variables.

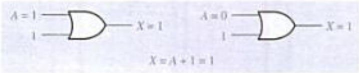
Rule 1. $A + 0 = A$ A variable ORed with 0 is always equal to the variable. If the input variable A is 1, the output variable X is 1, which is equal to A. If A is 0, the output is 0, which is also equal to A. This rule is illustrated in Figure 4–6, where the lower input is fixed at 0.

▶ **FIGURE 4-6**



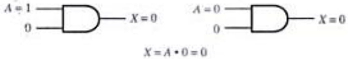
Rule 2. $A + 1 = 1$ A variable ORed with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input. This rule is illustrated in Figure 4–7, where the lower input is fixed at 1.

▶ **FIGURE 4-7**



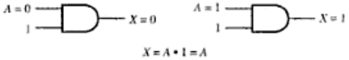
Rule 3. $A \cdot 0 = 0$ A variable ANDed with 0 is always equal to 0. Any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Figure 4–8, where the lower input is fixed at 0.

▶ **FIGURE 4-8**



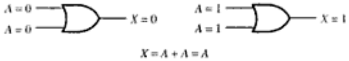
Rule 4. $A \cdot 1 = A$ A variable ANDed with 1 is always equal to the variable. If A is 0, the output of the AND gate is 0. If A is 1, the output of the AND gate is 1 because both inputs are now 1s. This rule is shown in Figure 4–9, where the lower input is fixed at 1.

FIGURE 4–9


$A = 0$
1
$X = 0$
$A = 1$
1
$X = 1$
$X = A \bullet 1 = A$

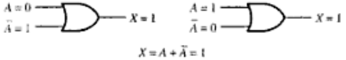
Rule 5. $A + A = A$ A variable ORed with itself is always equal to the variable. If A is 0, then $0 + 0 = 0$; and if A is 1, then $1 + 1 = 1$. This is shown in Figure 4–10, where both inputs are the same variable.

FIGURE 4–10


$A = 0$
$A = 0$
$X = 0$
$A = 1$
$A = 1$
$X = 1$
$X = A + A = A$

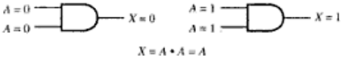
Rule 6. $A + \overline{A} = 1$ A variable ORed with its complement is always equal to 1. If A is 0, then $0 + \overline{0} = 0 + 1 = 1$. If A is 1, then $1 + \overline{1} = 1 + 0 = 1$. See Figure 4–11, where one input is the complement of the other.

FIGURE 4–11


$A = 0$
$\overline{A} = 1$
$X = 1$
$A = 1$
$\overline{A} = 0$
$X = 1$
$X = A + \overline{A} = 1$

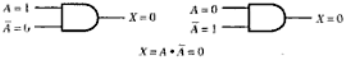
Rule 7. $A \cdot A = A$ A variable ANDed with itself is always equal to the variable. If $A = 0$, then $0 \cdot 0 = 0$; and if $A = 1$, then $1 \cdot 1 = 1$. Figure 4–12 illustrates this rule.

FIGURE 4–12


$A = 0$
$A = 0$
$X = 0$
$A = 1$
$A = 1$
$X = 1$
$X = A \bullet A = A$

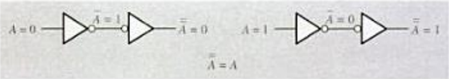
Rule 8. $A \cdot \overline{A} = 0$ A variable ANDed with its complement is always equal to 0. Either A or $\overline{A}$ will always be 0; and when a 0 is applied to the input of an AND gate, the output will be 0 also. Figure 4–13 illustrates this rule.

FIGURE 4–13


$A = 1$
$\overline{A} = 0$
$X = 0$
$A = 0$
$\overline{A} = 1$
$X = 0$
$X = A \bullet \overline{A} = 0$

Rule 9. $\bar{\bar{A}} = A$ The double complement of a variable is always equal to the variable. If you start with the variable A and complement (invert) it once, you get $\bar{A}$. If you then take $\bar{A}$ and complement (invert) it, you get A, which is the original variable. This rule is shown in Figure 4–14 using inverters.

▶ FIGURE 4–14



Rule 10. $A + AB = A$ This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

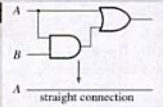
$$\begin{aligned} A + AB &= A(1 + B) && \text{Factoring (distributive law)} \\ &= A \cdot 1 && \text{Rule 2: } (1 + B) = 1 \\ &= A && \text{Rule 4: } A \cdot 1 = A \end{aligned}$$

The proof is shown in Table 4–2, which shows the truth table and the resulting logic circuit simplification.

▶ TABLE 4–2

A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal



Rule 11. $A + \bar{A}B = A + B$ This rule can be proved as follows:

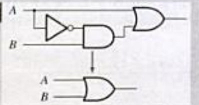
$$\begin{aligned} A + \bar{A}B &= (A + AB) + \bar{A}B && \text{Rule 10: } A = A + AB \\ &= (AA + AB) + \bar{A}B && \text{Rule 7: } A = AA \\ &= AA + AB + A\bar{A} + \bar{A}B && \text{Rule 8: adding } A\bar{A} = 0 \\ &= (A + \bar{A})(A + B) && \text{Factoring} \\ &= 1 \cdot (A + B) && \text{Rule 6: } A + \bar{A} = 1 \\ &= A + B && \text{Rule 4: drop the 1} \end{aligned}$$

The proof is shown in Table 4–3, which shows the truth table and the resulting logic circuit simplification.

▶ TABLE 4–3

A	B	$\bar{A}B$	$A + \bar{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

equal



Rule 12. $(A + B)(A + C) = A + BC$ This rule can be proved as follows:

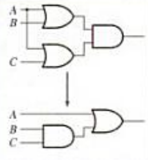
$$\begin{aligned}
(A + B)(A + C) &= AA + AC + AB + BC && \text{Distributive law}\\
&= A + AC + AB + BC && \text{Rule 7: } AA = A\\
&= A(1 + C) + AB + BC && \text{Factoring (distributive law)}\\
&= A \cdot 1 + AB + BC && \text{Rule 2: } 1 + C = 1\\
&= A(1 + B) + BC && \text{Factoring (distributive law)}\\
&= A \cdot 1 + BC && \text{Rule 2: } 1 + B = 1\\
&= A + BC && \text{Rule 4: } A \cdot 1 = A
\end{aligned}$$

The proof is shown in Table 4–4, which shows the truth table and the resulting logic circuit simplification.

▼ **TABLE 4–4**

A	B	C	$A + B$	$A + C$	$(A + B)(A + C)$	BC	$A + BC$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal



SECTION 4–2 REVIEW

1. Apply the associative law of addition to the expression $A + (B + C + D)$.
2. Apply the distributive law to the expression $A(B + C + D)$.

4–3 DEMORGAN'S THEOREMS

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms, DeMorgan's theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates, which were discussed in Chapter 3.

After completing this section, you should be able to

- State DeMorgan's theorems
- Relate DeMorgan's theorems to the equivalency of the NAND and negative-OR gates and to the equivalency of the NOR and negative-AND gates
- Apply DeMorgan's theorems to the simplification of Boolean expressions

One of DeMorgan's theorems is stated as follows:

The complement of a product of variables is equal to the sum of the complements of the variables.

Stated another way,

The complement of two or more variables ANDed is equivalent to the OR of the complements of the individual variables.

The formula for expressing this theorem for two variables is

Equation 4–6
$$\overline{XY} = \overline{X} + \overline{Y}$$

DeMorgan's second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

Stated another way,

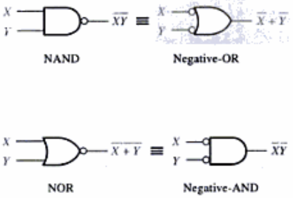
The complement of two or more variables ORed is equivalent to the AND of the complements of the individual variables.

The formula for expressing this theorem for two variables is

Equation 4–7
$$\overline{X + Y} = \overline{X}\,\overline{Y}$$

Figure 4–15 shows the gate equivalencies and truth tables for Equations 4–6 and 4–7.

FIGURE 4–15



Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X}+\overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Inputs		Output	
X	Y	$\overline{X+Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

As stated, DeMorgan's theorems also apply to expressions in which there are more than two variables. The following examples illustrate the application of DeMorgan's theorems to 3-variable and 4-variable expressions.

EXAMPLE 4–3

Apply DeMorgan's theorems to the expressions $\overline{XYZ}$ and $\overline{X + Y + Z}$.

Solution

$$\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$$

Supplementary Problem Apply DeMorgan's theorem to the expression $\overline{\overline{X} + \overline{Y} + \overline{Z}}$.

EXAMPLE 4–4

Apply DeMorgan's theorems to the expressions $\overline{WXYZ}$ and $\overline{W + X + Y + Z}$.

Solution

$$\overline{WXYZ} = \overline{W} + \overline{X} + \overline{Y} + \overline{Z}$$
$$\overline{W + X + Y + Z} = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$$

Supplementary Problem Apply DeMorgan's theorem to the expression $\overline{\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}}$.

Each variable in DeMorgan's theorems as stated in Equations 4–6 and 4–7 can also represent a combination of other variables. For example, X can be equal to the term $AB + C$, and Y can be equal to the term $A + BC$. So if you can apply DeMorgan's theorem for two variables as stated by $\overline{XY} = \overline{X} + \overline{Y}$ to the expression $\overline{(AB + C)(A + BC)}$, you get the following result:

$$\overline{(AB + C)(A + BC)} = \overline{(AB + C)} + \overline{(A + BC)}$$

Notice that in the preceding result you have two terms, $\overline{AB + C}$ and $\overline{A + BC}$, to each of which you can again apply DeMorgan's theorem $\overline{X + Y} = \overline{X}\,\overline{Y}$ individually, as follows:

$$\overline{(AB + C)} + \overline{(A + BC)} = (\overline{AB})\overline{C} + \overline{A}(\overline{BC})$$

Notice that you still have two terms in the expression to which DeMorgan's theorem can again be applied. These terms are $\overline{AB}$ and $\overline{BC}$. A final application of DeMorgan's theorem gives the following result:

$$(\overline{AB})\overline{C} + \overline{A}(\overline{BC}) = (\overline{A} + \overline{B})\overline{C} + \overline{A}(\overline{B} + \overline{C})$$

Although this result can be simplified further by the use of Boolean rules and laws, DeMorgan's theorems cannot be used any more.

Applying DeMorgan's Theorems

The following procedure illustrates the application of DeMorgan's theorems and Boolean algebra to the specific expression

$$\overline{\overline{A + B\overline{C}} + \overline{D(E + \overline{F})}}$$

Step 1. Identify the terms to which you can apply DeMorgan's theorems, and think of each term as a single variable. Let $\overline{A + B\overline{C}} = X$ and $\overline{D(E + \overline{F})} = Y$.

Step 2. Since $\overline{X + Y} = \overline{X}\,\overline{Y}$,

$$\overline{\overline{(A + B\overline{C})} + \overline{(D(E + \overline{F}))}} = \overline{\overline{(A + B\overline{C})}}\,\overline{\overline{(D(E + \overline{F}))}}$$

Step 3. Use rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the left term (this is not part of DeMorgan's theorem).

$$\overline{\overline{(A + B\overline{C})}}\,\overline{\overline{(D(E + \overline{F}))}} = (A + B\overline{C})\overline{(D(E + \overline{F}))}$$

Step 4. Applying DeMorgan's theorem to the second term,

$$(A + B\overline{C})\overline{(D(E + \overline{F}))} = (A + B\overline{C})(\overline{D} + \overline{\overline{(E + \overline{F})}})$$

Step 5. Use the rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the $E + \overline{F}$ part of the term.

$$(A + B\overline{C})(\overline{D} + \overline{\overline{E + \overline{F}}}) = (A + B\overline{C})(\overline{D} + E + \overline{F})$$

The following three examples will further illustrate how to use DeMorgan's theorems.

EXAMPLE 4–5

Apply DeMorgan's theorems to each of the following expressions:

(a) $\overline{(A + B + C)D}$ **(b)** $\overline{ABC + DEF}$ **(c)** $\overline{A\overline{B} + \overline{C}D + EF}$

Solution **(a)** Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

$$\overline{(A + B + C)D} = \overline{A + B + C} + \overline{D}$$

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

$$\overline{A + B + C} + \overline{D} = \overline{A}\,\overline{B}\,\overline{C} + \overline{D}$$

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X}\,\overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

$$(\overline{ABC})(\overline{DEF}) = (\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$$

(c) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B} + \overline{C}D + EF}$ is of the form $\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$ and can be rewritten as

$$\overline{A\overline{B} + \overline{C}D + EF} = (\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{A\overline{B}}$, $\overline{\overline{C}D}$, and $\overline{EF}$.

$$(\overline{A\overline{B}})(\overline{\overline{C}D})(\overline{EF}) = (\overline{A} + B)(C + \overline{D})(\overline{E} + \overline{F})$$

Supplementary Problem Apply DeMorgan's theorems to the expression $\overline{\overline{ABC} + D + E}$.

EXAMPLE 4–6

Apply DeMorgan's theorems to each expression:

(a) $\overline{\overline{(A + B)} + \overline{C}}$ **(b)** $\overline{(\overline{A} + B) + CD}$ **(c)** $\overline{(A + B)\overline{C}\,\overline{D} + E + \overline{F}}$

Solution **(a)** $\overline{\overline{(A + B)} + \overline{C}} = \overline{\overline{(A + B)}}\,\overline{\overline{C}} = (A + B)C$

(b) $\overline{(\overline{A} + B) + CD} = \overline{(\overline{A} + B)}\,\overline{CD} = (\overline{\overline{A}}\,\overline{B})(\overline{C} + \overline{D}) = A\overline{B}(\overline{C} + \overline{D})$

(c) $\overline{(A + B)\overline{C}\,\overline{D} + E + \overline{F}} = \overline{((A + B)\overline{C}\,\overline{D})}(\overline{E + \overline{F}}) = (\overline{A}\,\overline{B} + C + D)\overline{E}F$

Supplementary Problem Apply DeMorgan's theorems to the expression $\overline{\overline{A}B(C + \overline{D}) + E}$.

EXAMPLE 4–7

The Boolean expression for an exclusive-OR gate is $A\overline{B} + \overline{A}B$. With this as a starting point, use DeMorgan's theorems and any other rules or laws that are applicable to develop an expression for the exclusive-NOR gate.

Solution Start by complementing the exclusive-OR expression and then applying DeMorgan's theorems as follows:

$$\overline{A\overline{B} + \overline{A}B} = (\overline{A\overline{B}})(\overline{\overline{A}B}) = (\overline{A} + \overline{\overline{B}})(\overline{\overline{A}} + \overline{B}) = (\overline{A} + B)(A + \overline{B})$$

Next, apply the distributive law and rule 8 ($A \cdot \overline{A} = 0$).

$$(\overline{A} + B)(A + \overline{B}) = \overline{A}A + \overline{A}\,\overline{B} + AB + B\overline{B} = \overline{A}\,\overline{B} + AB$$

The final expression for the XNOR is $\overline{A}\,\overline{B} + AB$. Note that this expression equals 1 any time both variables are 0s or both variables are 1s.

Supplementary Problem Starting with the expression for a 4-input NAND gate, use DeMorgan's theorems to develop an expression for a 4-input negative-OR gate.

SECTION 4–3 REVIEW

1. Apply DeMorgan's theorems to the following expressions:

(a) $\overline{ABC} + (\overline{\overline{D} + E})$ (b) $\overline{(A + B)C}$ (c) $\overline{A + B + C} + \overline{\overline{D}E}$

4–4 BOOLEAN ANALYSIS OF LOGIC CIRCUITS

Boolean algebra provides a concise way to express the operation of a logic circuit formed by a combination of logic gates so that the output can be determined for various combinations of input values.

After completing this section, you should be able to

- Determine the Boolean expression for a combination of gates
- Evaluate the logic operation of a circuit from the Boolean expression
- Construct a truth table

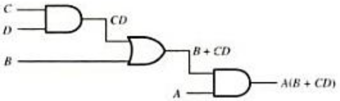
Boolean Expression for a Logic Circuit

To derive the Boolean expression for a given logic circuit, begin at the left-most inputs and work toward the final output, writing the expression for each gate. For the example circuit in Figure 4–16, the Boolean expression is determined as follows:

1. The expression for the left-most AND gate with inputs C and D is CD.
2. The output of the left-most AND gate is one of the inputs to the OR gate and B is the other input. Therefore, the expression for the OR gate is $B + CD$.
3. The output of the OR gate is one of the inputs to the right-most AND gate and A is the other input. Therefore, the expression for this AND gate is $A(B + CD)$, which is the final output expression for the entire circuit.

A logic circuit can always be described by a Boolean equation.

FIGURE 4–16



Constructing a Truth Table for a Logic Circuit

Once the Boolean expression for a given logic circuit has been determined, a truth table that shows the output for all possible values of the input variables can be developed. The procedure requires that you evaluate the Boolean expression for all possible combinations of values for the input variables. In the case of the circuit in Figure 4–16, there are four input variables (A, B, C, and D) and therefore, sixteen ($2^4 = 16$) combinations of values are possible. A logic circuit can always be described by a truth table.

Evaluating the Expression To evaluate the expression $A(B + CD)$, first find the values of the variables that make the expression equal to 1, using the rules for Boolean addition and multiplication. In this case, the expression equals 1 only if $A = 1$ and $B + CD = 1$ because

$$A(B + CD) = 1 \cdot 1 = 1$$

Now, determine when the $B + CD$ term equals 1. The term $B + CD = 1$ if either $B = 1$ or $CD = 1$ or if both B and CD equal 1 because

$$B + CD = 1 + 0 = 1$$
$$B + CD = 0 + 1 = 1$$
$$B + CD = 1 + 1 = 1$$

The term $CD = 1$ only if $C = 1$ and $D = 1$.

To summarize, the expression $A(B + CD) = 1$ when $A = 1$ and $B = 1$ regardless of the values of C and D or when $A = 1$ and $C = 1$ and $D = 1$ regardless of the value of B. The expression $A(B + CD) = 0$ for all other value combinations of the variables.

TABLE 4–5

Truth table for the logic circuit in Figure 4–16

INPUTS				OUTPUT
A	**B**	**C**	**D**	**A(B + CD)**
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Putting the Results in Truth Table Format The first step is to list the sixteen input variable combinations of 1s and 0s in a binary sequence as shown in Table 4–5. Next, place a 1 in the output column for each combination of input variables that was determined in the evaluation. Finally, place a 0 in the output column for all other combinations of input variables. These results are shown in the truth table in Table 4–5.

SECTION 4–4 REVIEW

1. Replace the AND gates with OR gates and the OR gate with an AND gate in Figure 4–16 and determine the Boolean expression for the output.
2. Construct a truth table for the circuit in Question 1.

4–5 SIMPLIFICATION USING BOOLEAN ALGEBRA

Many times, in the application of Boolean algebra, you have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently. The approach taken in this section is to use the basic laws, rules, and theorems of Boolean algebra to manipulate and simplify an expression. This method depends on a thorough knowledge of Boolean algebra and considerable practice in its application, not to mention a little ingenuity and cleverness.

After completing this section, you should be able to

- Apply the laws, rules, and theorems of Boolean algebra to simplify general expressions

A simplified Boolean expression uses the fewest gates possible to implement a given expression. Four examples follow to illustrate Boolean simplification step by step.

EXAMPLE 4–8

Using Boolean algebra techniques, simplify this expression:

$$AB + A(B + C) + B(B + C)$$

Solution The following is not necessarily the only approach.

Step 1. Apply the distributive law to the second and third terms in the expression, as follows:

$$AB + AB + AC + BB + BC$$

Step 2. Apply rule 7 ($BB = B$) to the fourth term.

$$AB + AB + AC + B + BC$$

Step 3. Apply rule 5 ($AB + AB = AB$) to the first two terms.

$$AB + AC + B + BC$$

Step 4. Apply rule 10 ($B + BC = B$) to the last two terms.

$$AB + AC + B$$

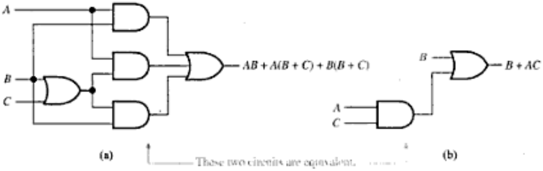
Step 5. Apply rule 10 ($AB + B = B$) to the first and third terms.

$$B + AC$$

At this point the expression is simplified as much as possible. Once you gain experience in applying Boolean algebra, you can often combine many individual steps.

Supplementary Problem Simplify the Boolean expression $A\overline{B} + A(\overline{B + C}) + B(\overline{B + C})$.

Figure 4–17 shows that the simplification process in Example 4–8 has significantly reduced the number of logic gates required to implement the expression. Part (a) shows that five gates are required to implement the expression in its original form; however, only two gates are needed for the simplified expression, shown in part (b). It is important to realize that these two gate circuits are equivalent. That is, for any combination of levels on the *A*, *B*, and *C* inputs, you get the same output from either circuit. In general, simplification leads to fewer gates for the same function.



▲ **FIGURE 4–17**

EXAMPLE 4–9

Simplify the following Boolean expression:

$$[A\overline{B}(C + BD) + \overline{A}\,\overline{B}]C$$

Note that brackets and parentheses mean the same thing: the term inside is multiplied (ANDed) with the term outside.

Solution **Step 1.** **Apply the distributive law to the terms within the brackets.**

$$(A\overline{B}C + A\overline{B}BD + \overline{A}\,\overline{B})C$$

Step 2. **Apply rule 8 ($\overline{B}B = 0$) to the second term within the parentheses.**

$$(A\overline{B}C + A \cdot 0 \cdot D + \overline{A}\,\overline{B})C$$

Step 3. **Apply rule 3 ($A \cdot 0 \cdot D = 0$) to the second term within the parentheses.**

$$(A\overline{B}C + 0 + \overline{A}\,\overline{B})C$$

Step 4. **Apply rule 1(drop the 0) within the parentheses.**

$$(A\overline{B}C + \overline{A}\,\overline{B})C$$

Step 5. **Apply the distributive law.**

$$A\overline{B}CC + \overline{A}\,\overline{B}C$$

Step 6. Apply rule 7 ($CC = C$) to the first term.

$$A\overline{B}C + \overline{A}\,\overline{B}C$$

Step 7. Factor out $\overline{B}C$.

$$\overline{B}C(A + \overline{A})$$

Step 8. Apply rule 6 ($A + \overline{A} = 1$).

$$\overline{B}C \cdot 1$$

Step 9. Apply rule 4 (drop the 1).

$$\overline{B}C$$

Supplementary Problem Simplify the Boolean expression $[AB(C + \overline{BD}) + \overline{AB}]CD$.

EXAMPLE 4–10

Simplify the following Boolean expression:

$$\overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + ABC$$

Solution **Step 1.** Factor BC out of the first and last terms.

$$BC(\overline{A} + A) + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C$$

Step 2. Apply rule 6 ($\overline{A} + A = 1$) to the term in parentheses, and factor $A\overline{B}$ from the second and last terms.

$$BC \cdot 1 + A\overline{B}(\overline{C} + C) + \overline{A}\,\overline{B}\,\overline{C}$$

Step 3. Apply rule 4 (drop the 1) to the first term and rule 6 ($\overline{C} + C = 1$) to the term in parentheses.

$$BC + A\overline{B} \cdot 1 + \overline{A}\,\overline{B}\,\overline{C}$$

Step 4. Apply rule 4 (drop the 1) to the second term.

$$BC + A\overline{B} + \overline{A}\,\overline{B}\,\overline{C}$$

Step 5. Factor $\overline{B}$ from the second and third terms.

$$BC + \overline{B}(A + \overline{A}\,\overline{C})$$

Step 6. Apply rule 11 ($A + \overline{A}\,\overline{C} = A + \overline{C}$) to the term in parentheses.

$$BC + \overline{B}(A + \overline{C})$$

Step 7. Use the distributive and commutative laws to get the following expression:

$$BC + A\overline{B} + \overline{B}\,\overline{C}$$

Supplementary Problem Simplify the Boolean expression $AB\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}\,\overline{C}$.

EXAMPLE 4–11

Simplify the following Boolean expression:

$$\overline{AB + AC} + \bar{A}\bar{B}C$$

Solution **Step 1.** Apply DeMorgan's theorem to the first term.

$$(\overline{AB})(\overline{AC}) + \bar{A}\bar{B}C$$

Step 2. Apply DeMorgan's theorem to each term in parentheses.

$$(\bar{A} + \bar{B})(\bar{A} + \bar{C}) + \bar{A}\bar{B}C$$

Step 3. Apply the distributive law to the two terms in parentheses.

$$\bar{A}\bar{A} + \bar{A}\bar{C} + \bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{B}C$$

Step 4. Apply rule 7 ($\bar{A}\bar{A} = \bar{A}$) to the first term, and apply rule 10 [$\bar{A}\bar{B} + \bar{A}\bar{B}C = \bar{A}\bar{B}(1 + C) = \bar{A}\bar{B}$] to the third and last terms.

$$\bar{A} + \bar{A}\bar{C} + \bar{A}\bar{B} + \bar{B}\bar{C}$$

Step 5. Apply rule 10 [$\bar{A} + \bar{A}\bar{C} = \bar{A}(1 + \bar{C}) = \bar{A}$] to the first and second terms.

$$\bar{A} + \bar{A}\bar{B} + \bar{B}\bar{C}$$

Step 6. Apply rule 10 [$\bar{A} + \bar{A}\bar{B} = \bar{A}(1 + \bar{B}) = \bar{A}$] to the first and second terms.

$$\bar{A} + \bar{B}\bar{C}$$

Supplementary Problem Simplify the Boolean expression $\overline{AB} + \overline{AC} + \bar{A}\bar{B}\bar{C}$.

SECTION 4–5 REVIEW

1. *Simplify the following Boolean expressions if possible:*

   (a) $A + AB + A\bar{B}C$ (b) $(\bar{A} + B)C + ABC$ (c) $A\bar{B}C(BD + CDE) + A\bar{C}$

2. Implement each expression in Question 1 as originally stated with the appropriate logic gates. Then implement the simplified expression, and compare the number of gates.

4–6 STANDARD FORMS OF BOOLEAN EXPRESSIONS

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of-sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

After completing this section, you should be able to

■ Identify a sum-of-products expression ■ Determine the domain of a Boolean expression ■ Convert any sum-of-products expression to a standard form ■ Evaluate a standard sum-of-products expression in terms of binary values ■ Identify a product-of-sums expression ■ Convert any product-of-sums expression to a standard form ■ Evaluate a standard product-of-sums expression in terms of binary values ■ Convert from one standard form to the other

The Sum-of-Products (SOP) Form

A product term was defined in Section 4–1 as a term consisting of the product (Boolean multiplication) of literals (variables or their complements). When two or more product terms are summed by Boolean addition, the resulting expression is a **sum-of-products (SOP)**. Some examples are

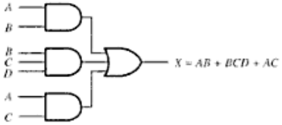
$$AB + ABC \qquad ABC + CDE + \overline{B}C\overline{D} \qquad \overline{A}B + \overline{A}B\overline{C} + AC$$

Also, an SOP expression can contain a single-variable term, as in $A + \overline{A}\overline{B}C + BC\overline{D}$. Refer to the simplification examples in the last section, and you will see that each of the final expressions was either a single product term or in SOP form. In an SOP expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar. For example, an SOP expression can have the term $\overline{A}\,\overline{B}\,\overline{C}$ but not $\overline{ABC}$.

Domain of a Boolean Expression The **domain** of a general Boolean expression is the set of variables contained in the expression in either complemented or uncomplemented form. For example, the domain of the expression $\overline{A}B + A\overline{B}C$ is the set of variables A, B, C and the domain of the expression $AB\overline{C} + C\overline{D}E + \overline{B}C\overline{D}$ is the set of variables A, B, C, D, E.

Implementation of an SOP Expression Implementing an SOP expression simply requires ORing the outputs of two or more AND gates. A product term is produced by an AND operation, and the sum (addition) of two or more product terms is produced by an OR operation. Therefore, an SOP expression can be implemented by AND-OR logic in which the outputs of a number (equal to the number of product terms in the expression) of AND gates connect to the inputs of an OR gate, as shown in Figure 4–18 for the expression $AB + BCD + AC$. The output X of the OR gate equals the SOP expression.

▶ **FIGURE 4–18**



Conversion of a General Expression to SOP Form

Any logic expression can be changed into SOP form by applying Boolean algebra techniques. For example, the expression $A(B + CD)$ can be converted to SOP form by applying the distributive law:

$$A(B + CD) = AB + ACD$$

EXAMPLE 4–12

Convert each of the following Boolean expressions to SOP form:

(a) $AB + B(CD + EF)$ **(b)** $(A + B)(B + C + D)$ **(c)** $\overline{(\overline{A + B}) + C}$

Solution **(a)** $AB + B(CD + EF) = AB + BCD + BEF$

(b) $(A + B)(B + C + D) = AB + AC + AD + BB + BC + BD$

(c) $\overline{(\overline{A + B}) + C} = (\overline{\overline{A + B}})\overline{C} = (A + B)\overline{C} = A\overline{C} + B\overline{C}$

Supplementary Problem Convert $\overline{A}B\overline{C} + (A + \overline{B})(B + \overline{C} + A\overline{B})$ to SOP form.

The Standard SOP Form

So far, you have seen SOP expressions in which some of the product terms do not contain all of the variables in the domain of the expression. For example, the expression $\overline{A}B\overline{C} + A\overline{B}D + \overline{A}B\overline{C}D$ has a domain made up of the variables A, B, C, and D. However, notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or $\overline{D}$ is missing from the first term and C or $\overline{C}$ is missing from the second term.

A standard SOP expression is one in which *all* the variables in the domain appear in each product term in the expression. For example, $A\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + AB\overline{C}\,\overline{D}$ is a standard SOP expression. Standard SOP expressions are important in constructing truth tables, covered in Section 4–7, and in the Karnaugh map simplification method, which is covered in Section 4–8. Any nonstandard SOP expression (referred to simply as SOP) can be converted to the standard form using Boolean algebra.

Converting Product Terms to Standard SOP Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard form using Boolean algebra rule 6 ($A + \overline{A} = 1$) from Table 4–1: A variable added to its complement equals 1.

Step 1. Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As you know, you can multiply anything by 1 without changing its value.

Step 2. Repeat Step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable, as Example 4–13 shows.

EXAMPLE 4–13

Convert the following Boolean expression into standard SOP form:

$$A\overline{B}C + \overline{A}\,\overline{B} + AB\overline{C}D$$

Solution The domain of this SOP expression is A, B, C, D. Take one term at a time. The first term, $A\overline{B}C$, is missing variable D or $\overline{D}$, so multiply the first term by $D + \overline{D}$ as follows:

$$A\overline{B}C = A\overline{B}C(D + \overline{D}) = A\overline{B}CD + A\overline{B}C\overline{D}$$

In this case, two standard product terms are the result.

The second term, $\overline{A}\,\overline{B}$, is missing variables C or $\overline{C}$ and D or $\overline{D}$, so first multiply the second term by $C + \overline{C}$ as follows:

$$\overline{A}\,\overline{B} = \overline{A}\,\overline{B}(C + \overline{C}) = \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C}$$

The two resulting terms are missing variable D or $\overline{D}$, so multiply both terms by $D + \overline{D}$ as follows:

$$\begin{aligned}\overline{A}\,\overline{B} &= \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} = \overline{A}\,\overline{B}C(D + \overline{D}) + \overline{A}\,\overline{B}\,\overline{C}(D + \overline{D}) \\ &= \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\end{aligned}$$

In this case, four standard product terms are the result.

The third term, $AB\overline{C}D$, is already in standard form. The complete standard SOP form of the original expression is as follows:

$$A\overline{B}C + \overline{A}\,\overline{B} + AB\overline{C}D = A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}D$$

Supplementary Problem Convert the expression $W\overline{X}Y + \overline{X}Y\overline{Z} + WX\overline{Y}$ to standard SOP form.

Binary Representation of a Standard Product Term A standard product term is equal to 1 for only one combination of variable values. For example, the product term $A\overline{B}C\overline{D}$ is equal to 1 when $A = 1$, $B = 0$, $C = 1$, $D = 0$, as shown below, and is 0 for all other combinations of values for the variables.

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

In this case, the product term has a binary value of 1010 (decimal ten).

Remember, a product term is implemented with an AND gate whose output is 1 only if each of its inputs is 1. Inverters are used to produce the complements of the variables as required.

An SOP expression is equal to 1 only if one or more of the product terms in the expression is equal to 1.

EXAMPLE 4–14

Determine the binary values for which the following standard SOP expression is equal to 1:

$$ABCD + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Solution The term $ABCD$ is equal to 1 when $A = 1$, $B = 1$, $C = 1$, and $D = 1$.

$$ABCD = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The term $A\overline{B}\,\overline{C}D$ is equal to 1 when $A = 1$, $B = 0$, $C = 0$, and $D = 1$.

$$A\overline{B}\,\overline{C}D = 1 \cdot \overline{0} \cdot \overline{0} \cdot 1 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The term $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ is equal to 1 when $A = 0$, $B = 0$, $C = 0$, and $D = 0$.

$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = \overline{0} \cdot \overline{0} \cdot \overline{0} \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

The SOP expression equals 1 when any or all of the three product terms is 1.

Supplementary Problem Determine the binary values for which the following SOP expression is equal to 1:

$$\overline{X}YZ + X\overline{Y}Z + XY\overline{Z} + \overline{X}Y\overline{Z} + XYZ$$

Is this a standard SOP expression?

The Product-of-Sums (POS) Form

A sum term was defined in Section 4–1 as a term consisting of the sum (Boolean addition) of literals (variables or their complements). When two or more sum terms are multiplied, the resulting expression is a **product-of-sums** (POS). Some examples are:

$$(\overline{A} + B)(A + \overline{B} + C)$$

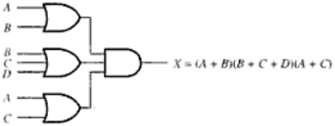
$$(\overline{A} + \overline{B} + \overline{C})(C + \overline{D} + E)(\overline{B} + C + D)$$

$$(A + B)(A + \overline{B} + C)(\overline{A} + C)$$

A POS expression can contain a single-variable term, as in $\overline{A}(A + \overline{B} + C)(\overline{B} + \overline{C} + D)$. In a POS expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar. For example, a POS expression can have the term $\overline{A} + \overline{B} + \overline{C}$ but not $\overline{A + B + C}$.

Implementation of a POS Expression Implementing a POS expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation, and the product of two or more sum terms is produced by an AND operation. Therefore, a POS expression can be implemented by logic in which the outputs of a number (equal to the number of sum terms in the expression) of OR gates connect to the inputs of an AND gate, as Figure 4–19 shows for the expression $(A + B)(B + C + D)(A + C)$. The output X of the AND gate equals the POS expression.

▶ FIGURE 4–19



The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression. For example, the expression

$$(A + \overline{B} + C)(A + B + \overline{D})(A + \overline{B} + \overline{C} + D)$$

has a domain made up of the variables A, B, C, and D. Notice that the complete set of variables in the domain is not represented in the first two terms of the expression; that is, D or $\overline{D}$ is missing from the first term and C or $\overline{C}$ is missing from the second term.

A standard POS expression is one in which *all* the variables in the domain appear in each sum term in the expression. For example,

$$(\overline{A} + \overline{B} + \overline{C} + \overline{D})(A + \overline{B} + C + D)(A + B + \overline{C} + D)$$

is a standard POS expression. Any nonstandard POS expression (referred to simply as POS) can be converted to the standard form using Boolean algebra.

Converting a Sum Term to Standard POS Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard POS expression is converted into standard form using Boolean algebra rule 8 ($A \cdot \overline{A} = 0$) from Table 4–1: A variable multiplied by its complement equals 0.

Step 1. Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms. As you know, you can add 0 to anything without changing its value.

Step 2. Apply rule 12 from Table 4–1: $A + BC = (A + B)(A + C)$

Step 3. Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or uncomplemented form.

EXAMPLE 4–15

Convert the following Boolean expression into standard POS form:

$$(A + \overline{B} + C)(\overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)$$

Solution The domain of this POS expression is A, B, C, D. Take one term at a time. The first term, $A + \overline{B} + C$, is missing variable D or $\overline{D}$, so add $D\overline{D}$ and apply rule 12 as follows:

$$A + \overline{B} + C = A + \overline{B} + C + D\overline{D} = (A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})$$

The second term, $\overline{B} + C + \overline{D}$, is missing variable A or $\overline{A}$, so add $A\overline{A}$ and apply rule 12 as follows:

$$\overline{B} + C + \overline{D} = \overline{B} + C + \overline{D} + A\overline{A} = (A + \overline{B} + C + \overline{D})(\overline{A} + \overline{B} + C + \overline{D})$$

The third term, $A + \overline{B} + \overline{C} + D$, is already in standard form. The standard POS form of the original expression is as follows:

$$(A + \overline{B} + C)(\overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D) =$$
$$(A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + C + \overline{D})(\overline{A} + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)$$

Supplementary Problem Convert the expression $(A + \overline{B})(B + C)$ to standard POS form.

Binary Representation of a Standard Sum Term A standard sum term is equal to 0 for only one combination of variable values. For example, the sum term $A + \overline{B} + C + \overline{D}$ is 0 when $A = 0$, $B = 1$, $C = 0$, and $D = 1$, as shown below, and is 1 for all other combinations of values for the variables.

$$A + \overline{B} + C + \overline{D} = 0 + \overline{1} + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

In this case, the sum term has a binary value of 0101 (decimal 5). Remember, a sum term is implemented with an OR gate whose output is 0 only if each of its inputs is 0. Inverters are used to produce the complements of the variables as required.

A POS expression is equal to 0 only if one or more of the sum terms in the expression is equal to 0.

EXAMPLE 4–16

Determine the binary values of the variables for which the following standard POS expression is equal to 0:

$$(A + B + C + D)(A + \overline{B} + \overline{C} + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D})$$

Solution The term $A + B + C + D$ is equal to 0 when $A = 0$, $B = 0$, $C = 0$, and $D = 0$.

$$A + B + C + D = 0 + 0 + 0 + 0 = 0$$

The term $A + \overline{B} + \overline{C} + D$ is equal to 0 when $A = 0$, $B = 1$, $C = 1$, and $D = 0$.

$$A + \overline{B} + \overline{C} + D = 0 + \overline{1} + \overline{1} + 0 = 0 + 0 + 0 + 0 = 0$$

The term $\overline{A} + \overline{B} + \overline{C} + \overline{D}$ is equal to 0 when $A = 1$, $B = 1$, $C = 1$, and $D = 1$.

$$\overline{A} + \overline{B} + \overline{C} + \overline{D} = \overline{1} + \overline{1} + \overline{1} + \overline{1} = 0 + 0 + 0 + 0 = 0$$

The POS expression equals 0 when any of the three sum terms equals 0.

Supplementary Problem Determine the binary values for which the following POS expression is equal to 0:

$$(X + \overline{Y} + Z)(\overline{X} + Y + Z)(X + Y + \overline{Z})(\overline{X} + \overline{Y} + \overline{Z})(X + \overline{Y} + \overline{Z})$$

Is this a standard POS expression?

Converting Standard SOP to Standard POS

The binary values of the product terms in a given standard SOP expression are not present in the equivalent standard POS expression. Also, the binary values that are not represented in the SOP expression are present in the equivalent POS expression. Therefore, to convert from standard SOP to standard POS, the following steps are taken:

Step 1. Evaluate each product term in the SOP expression. That is, determine the binary numbers that represent the product terms.

Step 2. Determine all of the binary numbers not included in the evaluation in Step 1.

Step 3. Write the equivalent sum term for each binary number from Step 2 and express in POS form.

Using a similar procedure, you can go from POS to SOP.

EXAMPLE 4–17

Convert the following SOP expression to an equivalent POS expression:

$$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + ABC$$

Solution The evaluation is as follows:

$$000 + 010 + 011 + 101 + 111$$

Since there are three variables in the domain of this expression, there are a total of eight (2^3) possible combinations. The SOP expression contains five of these combinations, so the POS must contain the other three which are 001, 100, and 110. Remember, these are the binary values that make the sum term 0. The equivalent POS expression is

$$(A + B + \overline{C})(\overline{A} + B + C)(\overline{A} + \overline{B} + C)$$

Supplementary Problem Verify that the SOP and POS expressions in this example are equivalent by substituting binary values into each.

SECTION 4–6 REVIEW

1. Identify each of the following expressions as SOP, standard SOP, POS, or standard POS:
   (a) $AB + \overline{A}BD + \overline{A}\,\overline{C}\,\overline{D}$
   (b) $(A + \overline{B} + C)(A + B + \overline{C})$
   (c) $\overline{A}BC + AB\overline{C}$
   (d) $A(A + \overline{C})(A + B)$
2. Convert each SOP expression in Question 1 to standard form.
3. Convert each POS expression in Question 1 to standard form.

4–7 BOOLEAN EXPRESSIONS AND TRUTH TABLES

All standard Boolean expressions can be easily converted into truth table format using binary values for each term in the expression. The truth table is a common way of presenting, in a concise format, the logical operation of a circuit. Also, standard SOP or POS expressions can be determined from a truth table. You will find truth tables in data sheets and other literature related to the operation of digital circuits.

After completing this section, you should be able to

- Convert a standard SOP expression into truth table format
- Convert a standard POS expression into truth table format
- Derive a standard expression from a truth table
- Properly interpret truth table data

Converting SOP Expressions to Truth Table Format

Recall from Section 4–6 that an SOP expression is equal to 1 only if at least one of the product terms is equal to 1. A truth table is simply a list of the possible combinations of input variable values and the corresponding output values (1 or 0). For an expression with a domain of two variables, there are four different combinations of those variables ($2^2 = 4$). For an expression with a domain of three variables, there are eight different combinations of those variables ($2^3 = 8$). For an expression with a domain of four variables, there are sixteen different combinations of those variables ($2^4 = 16$), and so on.

The first step in constructing a truth table is to list all possible combinations of binary values of the variables in the expression. Next, convert the SOP expression to standard form if it is not already. Finally, place a 1 in the output column (X) for each binary value that makes the standard SOP expression a 1 and place a 0 for all the remaining binary values. This procedure is illustrated in Example 4–18.

EXAMPLE 4–18

Develop a truth table for the standard SOP expression $\overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + ABC$.

Solution There are three variables in the domain, so there are eight possible combinations of binary values of the variables as listed in the left three columns of Table 4–6. The binary values that make the product terms in the expressions equal to 1 are $\overline{A}\,\overline{B}C$: 001; $A\overline{B}\,\overline{C}$: 100; and ABC: 111. For each of these binary values, a 1 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 0 is placed in the output column.

TABLE 4–6

INPUTS			OUTPUT	
A	B	C	X	PRODUCT TERM
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Supplementary Problem Create a truth table for the standard SOP expression $\overline{A}B\overline{C} + A\overline{B}C$.

Converting POS Expressions to Truth Table Format

Recall that a POS expression is equal to 0 only if at least one of the sum terms is equal to 0. To construct a truth table from a POS expression, list all the possible combinations of binary values of the variables just as was done for the SOP expression. Next, convert the POS expression to standard form if it is not already. Finally, place a 0 in the output column (X) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values. This procedure is illustrated in Example 4–19.

EXAMPLE 4–19

Determine the truth table for the following standard POS expression:

$$(A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + C)$$

Solution There are three variables in the domain and the eight possible binary values are listed in the left three columns of Table 4–7. The binary values that make the sum terms in the expression equal to 0 are $A + B + C$: 000; $A + \overline{B} + C$: 010; $A + \overline{B} + \overline{C}$: 011; $\overline{A} + B + \overline{C}$: 101; and $\overline{A} + \overline{B} + C$: 110. For each of these binary values, a 0 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 1 is placed in the output column.

▶ **TABLE 4–7**

INPUTS			OUTPUT	
A	B	C	X	SUM TERM
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \overline{B} + C)$
0	1	1	0	$(A + \overline{B} + \overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A} + B + \overline{C})$
1	1	0	0	$(\overline{A} + \overline{B} + C)$
1	1	1	1	

Notice that the truth table in this example is the same as the one in Example 4–18. This means that the SOP expression in the previous example and the POS expression in this example are equivalent.

Supplementary Problem Develop a truth table for the following standard POS expression:

$$(A + \overline{B} + C)(A + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})$$

Determining Standard Expressions from a Truth Table

To determine the standard SOP expression represented by a truth table, list the binary values of the input variables for which the output is 1. Convert each binary value to the corresponding product term by replacing each 1 with the corresponding variable and each 0 with the corresponding variable complement. For example, the binary value 1010 is converted to a product term as follows:

$$1010 \longrightarrow A\overline{B}C\overline{D}$$

If you substitute, you can see that the product term is 1:

$$A\overline{B}C\overline{D} = 1 \cdot \overline{0} \cdot 1 \cdot \overline{0} = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

To determine the standard POS expression represented by a truth table, list the binary values for which the output is 0. Convert each binary value to the corresponding sum term by replacing each 1 with the corresponding variable complement and each 0 with the corresponding variable. For example, the binary value 1001 is converted to a sum term as follows:

$$1001 \longrightarrow \overline{A} + B + C + \overline{D}$$

If you substitute, you can see that the sum term is 0:

$$\overline{A} + B + C + \overline{D} = \overline{1} + 0 + 0 + \overline{1} = 0 + 0 + 0 + 0 = 0$$

EXAMPLE 4–20

From the truth table in Table 4–8, determine the standard SOP expression and the equivalent standard POS expression.

TABLE 4–8

INPUTS			OUTPUT
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Solution There are four 1s in the output column and the corresponding binary values are 011, 100, 110, and 111. These binary values are converted to product terms as follows:

$$011 \longrightarrow \overline{A}BC$$
$$100 \longrightarrow A\overline{B}\,\overline{C}$$
$$110 \longrightarrow AB\overline{C}$$
$$111 \longrightarrow ABC$$

The resulting standard SOP expression for the output X is

$$X = \overline{A}BC + A\overline{B}\,\overline{C} + AB\overline{C} + ABC$$

For the POS expression, the output is 0 for binary values 000, 001, 010, and 101. These binary values are converted to sum terms as follows:

$$000 \longrightarrow A + B + C$$
$$001 \longrightarrow A + B + \overline{C}$$
$$010 \longrightarrow A + \overline{B} + C$$
$$101 \longrightarrow \overline{A} + B + \overline{C}$$

The resulting standard POS expression for the output X is

$$X = (A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(\overline{A} + B + \overline{C})$$

Supplementary Problem By substitution of binary values, show that the SOP and the POS expressions derived in this example are equivalent; that is, for any binary value they should either both be 1 or both be 0, depending on the binary value.

SECTION 4–7 REVIEW

1. If a certain Boolean expression has a domain of five variables, how many binary values will be in its truth table?
2. In a certain truth table, the output is a 1 for the binary value 0110. Convert this binary value to the corresponding product term using variables W, X, Y, and Z.
3. In a certain truth table, the output is a 0 for the binary value 1100. Convert this binary value to the corresponding sum term using variables W, X, Y, and Z.

4–8 THE KARNAUGH MAP

The Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression. As you have seen, the effectiveness of algebraic simplification depends on your familiarity with all the laws, rules, and theorems of Boolean algebra and on your ability to apply them. The Karnaugh map, on the other hand, basically provides a "cookbook" method for simplification.

After completing this section, you should be able to

■ Construct a Karnaugh map for three or four variables ■ Determine the binary value of each cell in a Karnaugh map ■ Determine the standard product term represented by each cell in a Karnaugh map ■ Explain cell adjacency and identify adjacent cells

A **Karnaugh map** is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of **cells** in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four, and five variables, but we will discuss only 3-variable and 4-variable situations to illustrate the principles. Section 4–11 deals with five variables using a 32-cell Karnaugh map. Another method, which is beyond the scope of this book, called the Quine-McClusky method can be used for higher numbers of variables. A Karnaugh map is used to simply a Boolean expression.

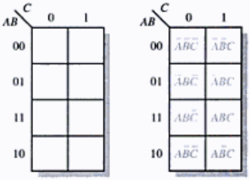
The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.

The 3-Variable Karnaugh Map

The 3-variable Karnaugh map is an array of eight cells, as shown in Figure 4–20(a). In this case, *A*, *B*, and *C* are used for the variables although other letters could be used. Binary values of *A* and *B* are along the left side (notice the sequence) and the values of *C* are across the top. The value of a given cell is the binary values of *A* and *B* at the left in the same row combined with the value of *C* at the top in the same column. For example, the cell in the upper left corner has a binary value of 000 and the cell in the lower right corner has a binary value of 101. Figure 4–20(b) shows the standard product terms that are represented by each cell in the Karnaugh map.

FIGURE 4–20

A 3-variable Karnaugh map


C
AB
0
1
00
01
11
10
$\overline{A}\,\overline{B}\,\overline{C}$
$\overline{A}\,\overline{B}C$
$\overline{A}B\overline{C}$
$\overline{A}BC$
$AB\overline{C}$
ABC
$A\overline{B}\,\overline{C}$
$A\overline{B}C$

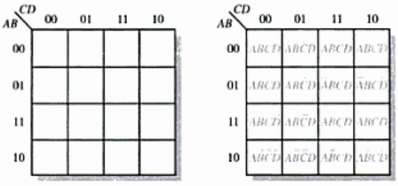
(a) (b)

The 4-Variable Karnaugh Map

The 4-variable Karnaugh map is an array of sixteen cells, as shown in Figure 4–21(a). Binary values of *A* and *B* are along the left side and the values of *C* and *D* are across the top. The value of a given cell is the binary values of *A* and *B* at the left in the same row combined with the binary values of *C* and *D* at the top in the same column. For example, the cell in the upper

▶ FIGURE 4–21

A 4-variable Karnaugh map



(a) (b)

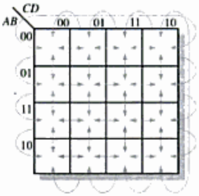
right corner has a binary value of 0010 and the cell in the lower right corner has a binary value of 1010. Figure 4–21(b) shows the standard product terms that are represented by each cell in the 4-variable Karnaugh map.

Cell Adjacency

The cells in a Karnaugh map are arranged so that there is only a single-variable change between adjacent cells. Adjacency is defined by a single-variable change. *Cells that differ by only one variable are adjacent.* For example, in the 3-variable map the 010 cell is adjacent to the 000 cell, the 011 cell, and the 110 cell. *Cells with values that differ by more than one variable are not adjacent.* For example, the 010 cell is not adjacent to the 001 cell, the 111 cell, the 100 cell, or the 101 cell.

Physically, each cell is adjacent to the cells that are immediately next to it on any of its four sides. A cell is not adjacent to the cells that diagonally touch any of its corners. Also, the cells in the top row are adjacent to the corresponding cells in the bottom row and the cells in the outer left column are adjacent to the corresponding cells in the outer right column. This is called "wrap-around" adjacency because you can think of the map as wrapping around from top to bottom to form a cylinder or from left to right to form a cylinder. Figure 4–22 illustrates the cell adjacencies with a 4-variable map, although the same rules for adjacency apply to Karnaugh maps with any number of cells.

▶ FIGURE 4–22



SECTION 4–8 REVIEW

1. In a 3-variable Karnaugh map, what is the binary value for the cell in each of the following locations:
   (a) upper left corner (b) lower right corner
   (c) lower left corner (d) upper right corner
2. What is the standard product term for each cell in Question 1 for variables *X*, *Y*, and *Z*?
3. Repeat Question 1 for a 4-variable map.
4. Repeat Question 2 for a 4-variable map using variables *W*, *X*, *Y*, and *Z*.

4–9 KARNAUGH MAP SOP MINIMIZATION

As stated in the last section, the Karnaugh map is used for simplifying Boolean expressions to their minimum form. A minimized SOP expression contains the fewest possible terms with the fewest possible variables per term. Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression.

After completing this section, you should be able to

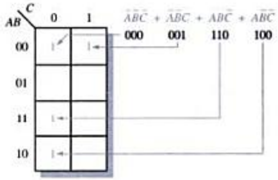
- Map a standard SOP expression on a Karnaugh map
- Combine the 1s on the map into maximum groups
- Determine the minimum product term for each group on the map
- Combine the minimum product terms to form a minimum SOP expression
- Convert a truth table into a Karnaugh map for simplification of the represented expression
- Use "don't care" conditions on a Karnaugh map

Mapping a Standard SOP Expression

For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression. Each 1 is placed in a cell corresponding to the value of a product term. For example, for the product term $A\overline{B}C$, a 1 goes in the 101 cell on a 3-variable map.

When an SOP expression is completely mapped, there will be a number of 1s on the Karnaugh map equal to the number of product terms in the standard SOP expression. The cells that do not have a 1 are the cells for which the expression is 0. Usually, when working with SOP expression, the 0s are left off the map. The following steps and the illustration in Figure 4–23 show the mapping process.

FIGURE 4–23


C
AB
0
1
00
01
11
10
1
1
1
1
$\overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + AB\overline{C} + A\overline{B}\overline{C}$
000 001 110 100

Step 1. Determine the binary value of each product term in the standard SOP expression. After some practice, you can usually do the evaluation of terms mentally.

Step 2. As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.

The folowing examples will further illustrate the mapping process.

EXAMPLE 4–21

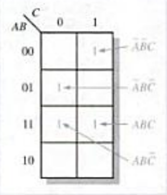
Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

Solution The expression is evaluated as shown below. A 1 is placed on the 3-variable Karnaugh map in Figure 4–24 for each standard product term in the expression.

$$\underset{001}{\overline{A}\overline{B}C} + \underset{010}{\overline{A}B\overline{C}} + \underset{110}{AB\overline{C}} + \underset{111}{ABC}$$

▶ FIGURE 4–24


C
AB
0
1
00
01
11
10
1
1
1
1
$\overline{A}\overline{B}C$
$\overline{A}B\overline{C}$
ABC
$AB\overline{C}$

Supplementary Problem Map the standard SOP expression $\overline{A}BC + A\overline{B}C + A\overline{B}\,\overline{C}$ on a Karnaugh map.

EXAMPLE 4–22

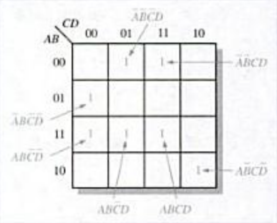
Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}C\overline{D}$$

Solution The expression is evaluated as shown below. A 1 is placed on the 4-variable Karnaugh map in Figure 4–25 for each standard product term in the expression.

$$\begin{array}{ccccccccccccc} \overline{A}\,\overline{B}CD & + & \overline{A}B\overline{C}\,\overline{D} & + & AB\overline{C}D & + & ABCD & + & AB\overline{C}\,\overline{D} & + & \overline{A}\,\overline{B}\,\overline{C}D & + & A\overline{B}C\overline{D} \\ 0011 & & 0100 & & 1101 & & 1111 & & 1100 & & 0001 & & 1010 \end{array}$$

▶ FIGURE 4–25


$\overline{A}\,\overline{B}\,\overline{C}D$
CD
AB
00
01
11
10
00
01
11
10
1
1
1
1
1
1
1
$\overline{A}\,\overline{B}CD$
$\overline{A}B\overline{C}\,\overline{D}$
$AB\overline{C}\,\overline{D}$
$A\overline{B}C\overline{D}$
$AB\overline{C}D$
$ABCD$

Supplementary Problem Map the following standard SOP expression on a Karnaugh map:

$$\overline{A}BC\overline{D} + ABC\overline{D} + AB\overline{C}\,\overline{D} + ABCD$$

Mapping a Nonstandard SOP Expression

A Boolean expression must first be in standard form before you use a Karnaugh map. If an expression is not in standard form, then it must be converted to standard form by the procedure covered in Section 4–6 or by numerical expansion. Since an expression should be evaluated before mapping anyway, numerical expansion is probably the most efficient approach.

Numerical Expansion of a Nonstandard Product Term Recall that a nonstandard product term has one or more missing variables. For example, assume that one of the product terms in a certain 3-variable SOP expression is $A\overline{B}$. This term can be expanded numerically to standard form as follows. First, write the binary value of the two variables and attach a 0 for the missing variable $\overline{C}$: 100. Next, write the binary value of the two variables and attach a 1 for the missing variable C: 101. The two resulting binary numbers are the values of the standard SOP terms $A\overline{B}\,\overline{C}$ and $A\overline{B}C$.

As another example, assume that one of the product terms in a 3-variable expression is B (remember that a single variable counts as a product term in an SOP expression). This term can be expanded numerically to standard form as follows. Write the binary value of the variable; then attach all possible values for the missing variables A and C as follows:

B
010
011
110
111

The four resulting binary numbers are the values of the standard SOP terms $\overline{A}B\overline{C}$, $\overline{A}BC$, $AB\overline{C}$, and ABC.

EXAMPLE 4–23

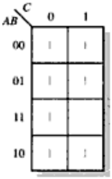
Map the following SOP expression on a Karnaugh map: $\overline{A} + A\overline{B} + AB\overline{C}$.

Solution The SOP expression is obviously not in standard form because each product term does not have three variables. The first term is missing two variables, the second term is missing one variable, and the third term is standard. First, expand the terms numerically as follows:

$\overline{A}$	$+ A\overline{B}$	$+ AB\overline{C}$
000	100	110
001	101	
010		
011		

Each of the resulting binary values is mapped by placing a 1 in the appropriate cell of the 3-variable Karnaugh map in Figure 4–26.

▶ FIGURE 4–26



Supplementary Problem Map the SOP expression $BC + \overline{A}\,\overline{C}$ on a Karnaugh map.

EXAMPLE 4–24

Map the following SOP expression on a Karnaugh map:

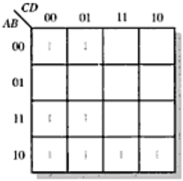
$$\overline{B}\,\overline{C} + A\overline{B} + AB\overline{C} + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}CD$$

Solution The SOP expression is obviously not in standard form because each product term does not have four variables. The first and second terms are both missing two variables, the third term is missing one variable, and the rest of the terms are standard. First, expand the terms by including all combinations of the missing variables numerically as follows:

$\overline{B}\,\overline{C}$	$+ A\overline{B}$	$+ AB\overline{C}$	$+ A\overline{B}C\overline{D}$	$+ \overline{A}\,\overline{B}\,\overline{C}D$	$+ A\overline{B}CD$
0000	1000	1100	1010	0001	1011
0001	1001	1101			
1000	1010				
1001	1011				

Each of the resulting binary values is mapped by placing a 1 in the appropriate cell of the 4-variable Karnaugh map in Figure 4–27. Notice that some of the values in the expanded expression are redundant.

▶ **FIGURE 4–27**



Supplementary Problem Map the expression $A + \overline{C}D + AC\overline{D} + \overline{A}BC\overline{D}$ on a Karnaugh map.

Karnaugh Map Simplification of SOP Expressions

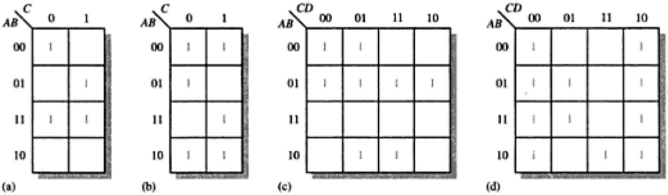
The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization.** After an SOP expression has been mapped, there are three steps in the process of obtaining a minimum SOP expression: grouping the 1s, determining the product term for each group, and summing the resulting product terms.

Grouping the 1s You can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s. The goal is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1, 2, 4, 8, or 16 cells, which are all powers of two. In the case of a 3-variable map, $2^3 = 8$ cells is the maximum group.
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.
3. Always include the largest possible number of 1s in a group in accordance with rule 1.
4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include non common 1s.

EXAMPLE 4–25

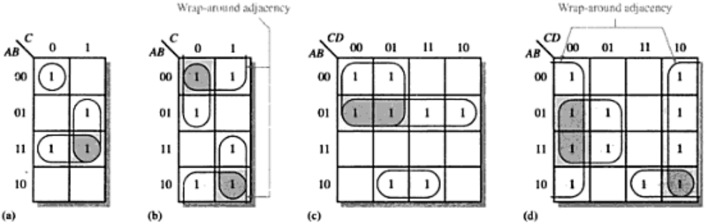
Group the 1s in each of the Karnaugh maps in Figure 4–28.



C
AB
0
1
00
01
11
10
CD
00
01
11
10
1
(a)
(b)
(c)
(d)

▲ **FIGURE 4–28**

Solution The groupings are shown in Figure 4–29. In some cases, there may be more than one way to group the 1s to form maximum groupings.



Wrap-around adjacency
Wrap-around adjacency
C
AB
0
1
00
01
11
10
CD
00
01
11
10
1
(a)
(b)
(c)
(d)

▲ **FIGURE 4–29**

Supplementary Problem Determine if there are other ways to group the 1s in Figure 4–29 to obtain a minimum number of maximum groupings.

Determining the Minimum SOP Expression from the Map When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining the resulting minimum SOP expression begins. The following rules are applied to find the minimum product terms and the minimum SOP expression:

1. Group the cells that have 1s. Each group of cells containing 1s creates one product term composed of all variables that occur in only one form (either uncomplemented or

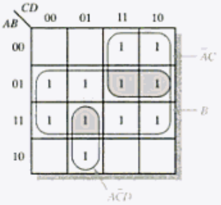
complemented) within the group. Variables that occur both uncomplemented and complemented within the group are eliminated. These are called *contradictory variables.*

2. Determine the minimum product term for each group.
   - a. For a 3-variable map:
     - (1) A 1-cell group yields a 3-variable product term
     - (2) A 2-cell group yields a 2-variable product term
     - (3) A 4-cell group yields a 1-variable term
     - (4) An 8-cell group yields a value of 1 for the expression
   - b. For a 4-variable map:
     - (1) A 1-cell group yields a 4-variable product term
     - (2) A 2-cell group yields a 3-variable product term
     - (3) A 4-cell group yields a 2-variable product term
     - (4) An 8-cell group yields a 1-variable term
     - (5) A 16-cell group yields a value of 1 for the expression
3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

EXAMPLE 4–26

Determine the product terms for the Karnaugh map in Figure 4–30 and write the resulting minimum SOP expression.

FIGURE 4–30



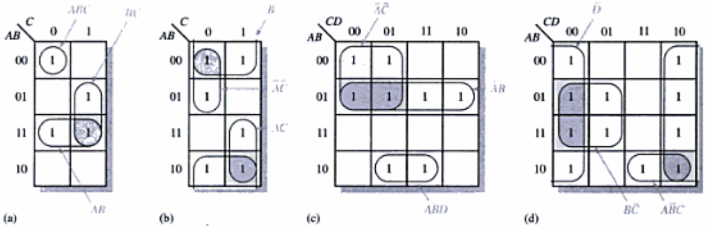
Solution In Figure 4–30, the product term for the 8-cell group is B because the cells within that group contain both A and $\overline{A}$, C and $\overline{C}$, and D and $\overline{D}$, so these variables are eliminated. The 4-cell group contains B, $\overline{B}$, D, and $\overline{D}$, leaving the variables $\overline{A}$ and C, which form the product term $\overline{A}C$. The 2-cell group contains B and $\overline{B}$, leaving variables A, $\overline{C}$, and D which form the product term $A\overline{C}D$. Notice how overlapping is used to maximize the size of the groups. The resulting minimum SOP expression is the sum of these product terms:

$$B + \overline{A}C + A\overline{C}D$$

Supplementary Problem For the Karnaugh map in Figure 4–30, add a 1 in the lower right cell (1010) and determine the resulting SOP expression.

EXAMPLE 4–27

Determine the product terms for each of the Karnaugh maps in Figure 4–31 and write the resulting minimum SOP expression.


$\overline{A}\overline{B}\overline{C}$
BC
AB
(a)
$\overline{B}$
$\overline{A}\overline{C}$
AC
(b)
$\overline{A}\overline{C}$
$\overline{A}B$
$A\overline{B}D$
(c)
$\overline{D}$
$B\overline{C}$
$A\overline{B}C$
(d)

▲ **FIGURE 4–31**

Solution The resulting minimum product term for each group is shown in Figure 4–31. The minimum SOP expressions for each of the Karnaugh maps in the figure are

(a) $AB + BC + \overline{A}\overline{B}\overline{C}$ **(b)** $\overline{B} + \overline{A}\overline{C} + AC$

(c) $\overline{A}B + \overline{A}\overline{C} + A\overline{B}D$ **(d)** $\overline{D} + A\overline{B}C + B\overline{C}$

Supplementary Problem For the Karnaugh map in Figure 4–31(d), add a 1 in the 0111 cell and determine the resulting SOP expression.

EXAMPLE 4–28

Use a Karnaugh map to minimize the following standard SOP expression:

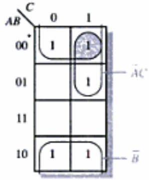
$$A\overline{B}C + \overline{A}BC + \overline{A}BC + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C}$$

Solution The binary values of the expression are

$$101 + 011 + 011 + 000 + 100$$

The standard SOP expression is mapped and the cells are grouped as shown in Figure 4–32.

▶ **FIGURE 4–32**


$\overline{A}C$
$\overline{B}$

Notice the "wrap around" 4-cell group that includes the top row and the bottom row of 1s. The remaining 1 is absorbed in an overlapping group of two cells. The group of four 1s produces a single variable term, $\overline{B}$. This is determined by observing that within the group, $\overline{B}$ is the only variable that does not change from cell to cell. The group of two 1s produces a 2-variable term $\overline{A}C$. This is determined by observing that within the group, $\overline{A}$ and C do not change from one cell to the next. The product term for each group is shown and the resulting minimum SOP expression is

$$\overline{B} + \overline{A}C$$

Keep in mind that this minimum expression is equivalent to the original standard expression.

Supplementary Problem Use a Karnaugh map to simplify the following standard SOP expression:

$$X\overline{Y}Z + XY\overline{Z} + \overline{X}YZ + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$

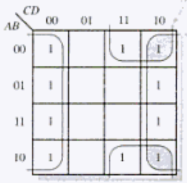
EXAMPLE 4–29

Use a Karnaugh map to minimize the following SOP expression:

$$\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + A\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$

Solution The first term $\overline{B}\,\overline{C}\,\overline{D}$ must be expanded into $A\overline{B}\,\overline{C}\,\overline{D}$ and $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ to get the standard SOP expression, which is then mapped; and the cells are grouped as shown in Figure 4–33.

▸ **FIGURE 4–33**



Notice that both groups exhibit "wrap around" adjacency. The group of eight is formed because the cells in the outer columns are adjacent. The group of four is formed to pick up the remaining two 1s because the top and bottom cells are adjacent. The product term for each group is shown and the resulting minimum SOP expression is

$$\overline{D} + \overline{B}C$$

Keep in mind that this minimum expression is equivalent to the original standard expression.

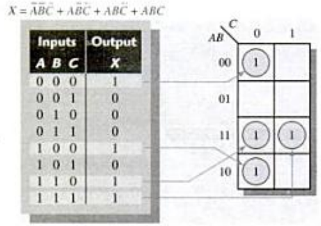
Supplementary Problem Use a Karnaugh map to simplify the following SOP expression:

$$\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}YZ + W\overline{X}\,\overline{Y}Z + \overline{W}YZ + W\overline{X}\,\overline{Y}\,\overline{Z}$$

Mapping Directly from a Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnaugh map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combinations. An example of a Boolean expression and its truth table representation is shown in Figure 4–34. Notice in the truth table that the output X is 1 for four different input variable combinations. The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to the values of the associated input variable combinations, as shown in Figure 4–34. In the figure you can see that the Boolean expression, the truth table, and the Karnaugh map are simply different ways to represent a logic function.

▶ **FIGURE 4–34**



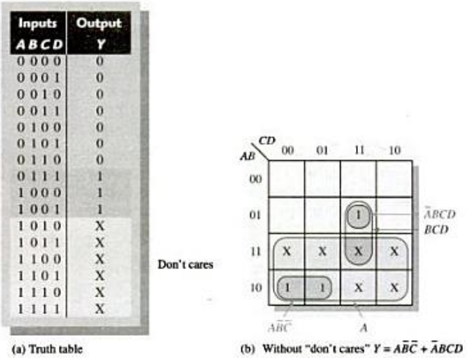
$X = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$

Inputs	Output
A B C	*X*
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	1

"Don't Care" Conditions

Sometimes a situation arises in which some input variable combinations are not allowed. For example, recall that in the BCD code covered in Chapter 2, there are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111. Since these unallowed states will never occur in an application involving the BCD code, they can be treated as **"don't care"** terms with respect to

▶ **FIGURE 4–35**



Inputs	Output
ABCD	*Y*
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1
1 0 1 0	X
1 0 1 1	X
1 1 0 0	X
1 1 0 1	X
1 1 1 0	X
1 1 1 1	X

(a) Truth table

(b) Without "don't cares" $Y = A\overline{B}\overline{C} + \overline{A}BCD$
With "don't cares" $Y = A + BCD$

their effect on the output. That is, for these "don't care" terms either a 1 or a 0 may be assigned to the output; it really does not matter since they will never occur.

The "don't care" terms can be used to advantage on the Karnaugh map. Figure 4–35 shows that for each "don't care" term, an X is placed in the cell. When grouping the 1s, the Xs can be treated as 1s to make a larger grouping or as 0s if they cannot be used to advantage. The larger a group, the simpler the resulting term will be.

The truth table in Figure 4–35(a) describes a logic function that has a 1 output only when the BCD code for 7, 8, or 9 is present on the inputs. If the "don't cares" are used as 1s, the resulting expression for the function is $A + BCD$, as indicated in part (b). If the "don't cares" are not used as 1s, the resulting expression is $A\overline{B}\overline{C} + \overline{A}BCD$. So you can see the advantage of using "don't care" terms to get the simplest expression.

SECTION 4–9 REVIEW

1. Lay out Karnaugh maps for three and four variables.
2. Group the 1s and write the simplified SOP expression for the Karnaugh map in Figure 4–24.
3. Write the original standard SOP expressions for each of the Karnaugh maps in Figure 4–31.

4–10 KARNAUGH MAP POS MINIMIZATION

In the last section, you studied the minimization of an SOP expression using a Karnaugh map. In this section, we will focus on POS expressions. The approaches are much the same except that with POS expressions, 0s representing the standard sum terms are placed on the Karnaugh map instead of 1s.

After completing this section, you should be able to

- Map a standard POS expression on a Karnaugh map
- Combine the 0s on the map into maximum groups
- Determine the minimum sum term for each group on the map
- Combine the minimum sum terms to form a minimum POS expression
- Use the Karnaugh map to convert between POS and SOP

Mapping a Standard POS Expression

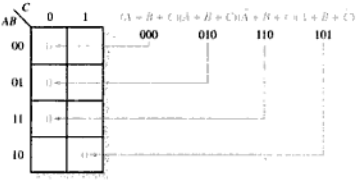
For a POS expression in standard form, a 0 is placed on the Karnaugh map for each sum term in the expression. Each 0 is placed in a cell corresponding to the value of a sum term. For example, for the sum term $A + \overline{B} + C$, a 0 goes in the 010 cell on a 3-variable map.

When a POS expression is completely mapped, there will be a number of 0s on the Karnaugh map equal to the number of sum terms in the standard POS expression. The cells that do not have a 0 are the cells for which the expression is 1. Usually, when working with POS expressions, the 1s are left off. The following steps and the illustration in Figure 4–36 show the mapping process.

Step 1. Determine the binary value of each sum term in the standard POS expression. This is the binary value that makes the term equal to 0.

Step 2. As each sum term is evaluated, place a 0 on the Karnaugh map in the corresponding cell.

FIGURE 4–36



Example 4–30 will further illustrate the mapping process.

EXAMPLE 4–30

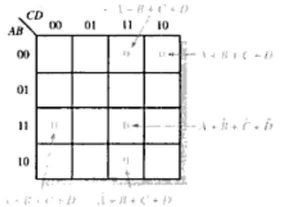
Map the following standard POS expression on a Karnaugh map:

$$(\overline{A} + \overline{B} + C + D)(\overline{A} + B + \overline{C} + \overline{D})(A + B + \overline{C} + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D})(A + B + \overline{C} + \overline{D})$$

Solution The expression is evaluated as shown below and a 0 is placed on the 4-variable Karnaugh map in Figure 4–37 for each standard sum term in the expression.

$$\underset{1100}{(\overline{A} + \overline{B} + C + D)}\underset{1011}{(\overline{A} + B + \overline{C} + \overline{D})}\underset{0010}{(A + B + \overline{C} + D)}\underset{1111}{(\overline{A} + \overline{B} + \overline{C} + \overline{D})}\underset{0011}{(A + B + \overline{C} + \overline{D})}$$

FIGURE 4–37



Supplementary Problem Map the following standard POS expression on a Karnaugh map:

$$(A + \overline{B} + \overline{C} + D)(A + B + C + \overline{D})(A + B + C + D)(\overline{A} + B + \overline{C} + D).$$

Karnaugh Map Simplification of POS Expressions

The process for minimizing a POS expression is basically the same as for an SOP expression except that you group 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms. The rules for grouping the 0s are the same as those for grouping the 1s that you learned in Section 4–9.

EXAMPLE 4–31

Use a Karnaugh map to minimize the following standard POS expression:

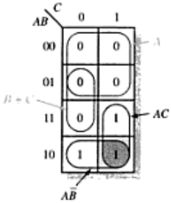
$$(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$

Solution The combinations of binary values of the expression are

$$(0 + 0 + 0)(0 + 0 + 1)(0 + 1 + 0)(0 + 1 + 1)(1 + 1 + 0)$$

The standard POS expression is mapped and the cells are grouped as shown in Figure 4–38.

▶ **FIGURE 4–38**



Notice how the 0 in the 110 cell is included into a 2-cell group by utilizing the 0 in the 4-cell group. The sum term for each blue group is shown in the figure and the resulting minimum POS expression is

$$A(\overline{B} + C)$$

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

Grouping the 1s as shown by the gray areas yields an SOP expression that is equivalent to grouping the 0s.

$$AC + A\overline{B} = A(\overline{B} + C)$$

Supplementary Problem Use a Karnaugh map to simplify the following standard POS expression:

$$(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(\overline{X} + \overline{Y} + Z)(\overline{X} + Y + Z)$$

EXAMPLE 4–32

Use a Karnaugh map to minimize the following POS expression:

$$(B + C + D)(A + B + \overline{C} + D)(\overline{A} + B + C + \overline{D})(A + \overline{B} + C + D)(\overline{A} + \overline{B} + C + D)$$

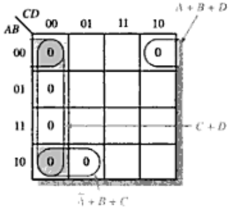
Solution The first term must be expanded into $\overline{A} + B + C + D$ and $A + B + C + D$ to get a standard POS expression, which is then mapped; and the cells are grouped as shown in

Figure 4–39. The sum term for each group is shown and the resulting minimum POS expression is

$$(C + D)(A + B + D)(\overline{A} + B + C)$$

Keep in mind that this minimum POS expression is equivalent to the original standard POS expression.

FIGURE 4–39



Supplementary Problem Use a Karnaugh map to simplify the following POS expression:

$$(W + \overline{X} + Y + \overline{Z})(W + X + Y + Z)(W + \overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + Z)$$

Converting Between POS and SOP Using the Karnaugh Map

When a POS expression is mapped, it can easily be converted to the equivalent SOP form directly from the Karnaugh map. Also, given a mapped SOP expression, an equivalent POS expression can be derived directly from the map. This provides a good way to compare both minimum forms of an expression to determine if one of them can be implemented with fewer gates than the other.

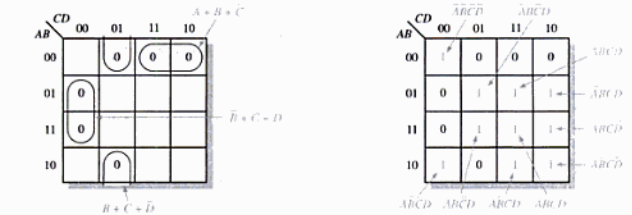
For a POS expression, all the cells that do not contain 0s contain 1s, from which the SOP expression is derived. Likewise, for an SOP expression, all the cells that do not contain 1s contain 0s, from which the POS expression is derived. Example 4–33 illustrates this conversion.

EXAMPLE 4–33

Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.

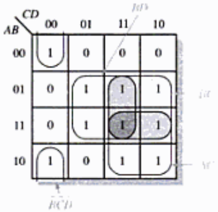
$$(\overline{A} + \overline{B} + C + D)(A + \overline{B} + C + D)(A + B + C + \overline{D})$$
$$(A + B + \overline{C} + \overline{D})(\overline{A} + B + C + \overline{D})(A + B + \overline{C} + D)$$

Solution The 0s for the standard POS expression are mapped and grouped to obtain the minimum POS expression in Figure 4–40(a). In Figure 4–40(b), 1s are added to the cells that do not contain 0s. From each cell containing a 1, a standard product term is obtained as indicated. These product terms form the standard SOP expression. In Figure 4–40(c), the 1s are grouped and a minimum SOP expression is obtained.


CD
AB
00
01
11
10
$A+B+\overline{C}$
$\overline{B}+C+D$
$B+C+\overline{D}$
$\overline{A}\overline{B}\overline{C}\overline{D}$
$\overline{A}B\overline{C}D$
$\overline{A}BCD$
$\overline{A}BC\overline{D}$
$ABC\overline{D}$
$A\overline{B}C\overline{D}$
$A\overline{B}\overline{C}\overline{D}$
$ABC\overline{D}$
$A\overline{B}CD$
$ABCD$

(a) Minimum POS: $(A+B+\overline{C})(\overline{B}+C+D)(B+C+\overline{D})$

(b) Standard SOP:
$\overline{A}\overline{B}\overline{C}\overline{D}+\overline{A}B\overline{C}D+\overline{A}BCD+\overline{A}BC\overline{D}+ABC\overline{D}+A\overline{B}C\overline{D}+$
$A\overline{B}\overline{C}\overline{D}+AB\overline{C}D+A\overline{B}CD+ABCD$


CD
AB
00
01
11
10
BD
BC
AC
$\overline{B}\overline{C}\overline{D}$

(c) Minimum SOP: $AC+BC+BD+\overline{B}\overline{C}\overline{D}$

▲ **FIGURE 4–40**

Supplementary Problem Use a Karnaugh map to convert the following expression to minimum SOP form:

$$(W+\overline{X}+Y+\overline{Z})(\overline{W}+X+\overline{Y}+\overline{Z})(\overline{W}+\overline{X}+\overline{Y}+Z)(\overline{W}+\overline{X}+\overline{Z})$$

SECTION 4–10 REVIEW

1. What is the difference in mapping a POS expression and an SOP expression?
2. What is the standard sum term for a 0 in cell 1011?
3. What is the standard product term for a 1 in cell 0010?

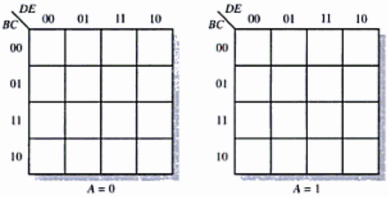
4–11 FIVE-VARIABLE KARNAUGH MAPS

Boolean functions with five variables can be simplified using a 32-cell Karnaugh map. Actually, two 4-variable maps (16 cells each) are used to construct a 5-variable map. You already know the cell adjacencies within each of the 4-variable maps and how to form groups of cells containing 1s to simplify an SOP expression. All you need to learn for five variables is the cell adjacencies between the two 4-variable maps and how to group those adjacent 1s.

After completing this section, you should be able to

- Determine cell adjacencies in a 5-variable map
- Form maximum cell groupings in a 5-variable map
- Minimize 5-variable Boolean expressions using the Karnaugh map

A Karnaugh map for five variables ($ABCDE$) can be constructed using two 4-variable maps with which you are already familiar. Each map contains 16 cells with all combinations of variables B, C, D, and E. One map is for $A = 0$ and the other is for $A = 1$, as shown in Figure 4–41.



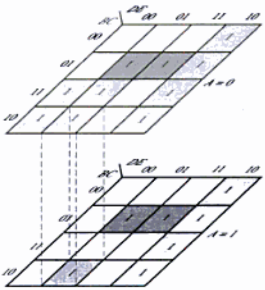
▲ **FIGURE 4–41**

Cell Adjacencies

You already know how to determine adjacent cells within the 4-variable map. The best way to visualize cell adjacencies between the two 16-cell maps is to imagine that the $A = 0$ map is placed on top of the $A = 1$ map. Each cell in the $A = 0$ map is adjacent to the cell directly below it in the $A = 1$ map.

To illustrate, an example with four groups is shown in Figure 4–42 with the maps in a 3-dimensional arrangement. The 1s in the yellow cells form an 8-bit group (four in the $A = 0$ map combined with four in the $A = 1$ map). The 1s in the orange cells form a 4-bit group. The 1s in the light red cells form a 4-bit group only in the $A = 0$ map. The 1 in the gray cell in the $A = 1$ map is grouped with the 1 in the lower right light red cell in the $A = 0$ map to form a 2-bit group.

► **FIGURE 4–42**



The Simplified Boolean Expression

The original SOP Boolean expression that is plotted on the Karnaugh map in Figure 4–42 contains seventeen 5-variable terms because there are seventeen 1s on the map. As you know, only the variables that do not change within a group remain in the expression for that group. The simplified expression taken from the map is developed as follows:

- The term for the yellow group is $D\overline{E}$.
- The term for the orange group is $\overline{B}CE$.
- The term for the light red group is $\overline{A}B\overline{D}$.
- The term for the gray cell grouped with the red cell is $B\overline{C}\overline{D}E$.

Combining these terms into the simplified SOP expression yields

$$X = D\overline{E} + \overline{B}CE + \overline{A}B\overline{D} + B\overline{C}\overline{D}E$$

EXAMPLE 4–34

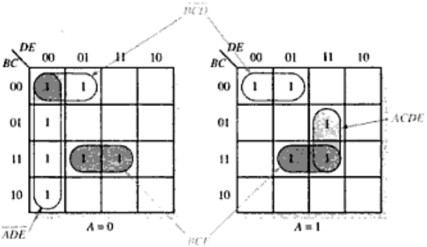
Use a Karnaugh map to minimize the following standard SOP 5-variable expression:

$$X = \overline{A}\overline{B}\overline{C}\overline{D}\overline{E} + \overline{A}\overline{B}C\overline{D}\overline{E} + \overline{A}BC\overline{D}\overline{E} + \overline{A}B\overline{C}\overline{D}\overline{E} + \overline{A}\overline{B}\overline{C}\overline{D}E + \overline{A}BC\overline{D}E$$
$$+ \overline{A}BCDE + A\overline{B}\overline{C}\overline{D}\overline{E} + A\overline{B}\overline{C}\overline{D}E + ABC\overline{D}E + ABCDE + A\overline{B}CDE$$

Solution The SOP expression is mapped in Figure 4–43 and the groupings and their corresponding terms are indicated. Combining the terms yields the following minimized SOP expression:

$$X + \overline{A}\overline{D}\overline{E} + \overline{B}\overline{C}\overline{D} + BCE + ACDE$$

▶ FIGURE 4–43


$\overline{B}\overline{C}\overline{D}$
DE
BC
00
01
11
10
1
A = 0
A = 1
$ACDE$
$\overline{A}\overline{D}\overline{E}$
BCE

Supplementary Problem Minimize the following expression:

$$Y = \overline{A}\overline{B}\overline{C}\overline{D}\overline{E} + \overline{A}\overline{B}C\overline{D}\overline{E} + \overline{A}BC\overline{D}\overline{E} + \overline{A}B\overline{C}\overline{D}\overline{E} + A\overline{B}\overline{C}\overline{D}\overline{E} + A\overline{B}C\overline{D}\overline{E} + ABC\overline{D}\overline{E} + AB\overline{C}\overline{D}\overline{E}$$
$$+ \overline{A}\overline{B}\overline{C}D\overline{E} + \overline{A}\overline{B}CD\overline{E} + \overline{A}BCD\overline{E} + \overline{A}B\overline{C}D\overline{E} + A\overline{B}\overline{C}D\overline{E} + A\overline{B}CD\overline{E} + ABCD\overline{E} + AB\overline{C}D\overline{E}$$

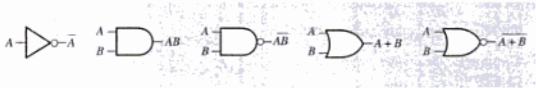
SECTION 4–11 REVIEW

1. **Why does a 5-variable Karnaugh map require 32 cells?**
2. **What is the expression represented by a 5-variable Karnaugh map in which each cell contains a 1?**

SUMMARY

- Gate symbols and Boolean expressions for the outputs of an inverter and 2-input gates are shown in Figure 4–44.

▶ **FIGURE 4–44**



- Commutative laws: $A + B = B + A$
  $AB = BA$
- Associative laws: $A + (B + C) = (A + B) + C$
  $A(BC) = (AB)C$
- Distributive law: $A(B + C) = AB + AC$
- Boolean rules:
  1. $A + 0 = A$
  2. $A + 1 = 1$
  3. $A \cdot 0 = 0$
  4. $A \cdot 1 = A$
  5. $A + A = A$
  6. $A + \bar{A} = 1$
  7. $A \cdot A = A$
  8. $A \cdot \bar{A} = 0$
  9. $\bar{\bar{A}} = A$
  10. $A + AB = A$
  11. $A + \bar{A}B = A + B$
  12. $(A + B)(A + C) = A + BC$
- DeMorgan's theorems:

  1. The complement of a product is equal to the sum of the complements of the terms in the product.

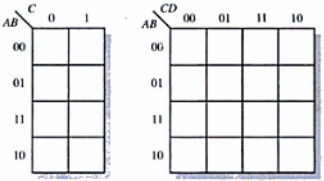
  $$\overline{XY} = \bar{X} + \bar{Y}$$

  2. The complement of a sum is equal to the product of the complements of the terms in the sum.

  $$\overline{X + Y} = \bar{X}\bar{Y}$$

- Karnaugh maps for 3 and 4 variables are shown in Figure 4–45. A 5-variable map is formed from two 4-variable maps.

▶ **FIGURE 4–45**



3-variable 4-variable

SELF-TEST

Answers are at the end of the chapter.

1. The complement of a variable is always

(a) 0 **(b)** 1 **(c)** equal to the variable **(d)** the inverse of the variable

2. The Boolean expression $A + \bar{B} + C$ is

(a) a sum term **(b)** a literal term **(c)** a product term **(d)** a complemented term

3. The Boolean expression $A\overline{B}C\overline{D}$ is

(a) a sum term **(b)** a product term **(c)** a literal term **(d)** always 1

4. The domain of the expression $A\overline{B}CD + A\overline{B} + \overline{C}D + B$ is

(a) A and D **(b)** B only **(c)** A, B, C, and D **(d)** none of these

5. According to the commutative law of addition,

(a) $AB = BA$ **(b)** $A = A + A$

(c) $A + (B + C) = (A + B) + C$ **(d)** $A + B = B + A$

6. According to the associative law of multiplication,

(a) $B = BB$ **(b)** $A(BC) = (AB)C$ **(c)** $A + B = B + A$ **(d)** $B + B(B + 0)$

7. According to the distributive law,

(a) $A(B + C) = AB + AC$ **(b)** $A(BC) = ABC$ **(c)** $A(A + 1) = A$ **(d)** $A + AB = A$

8. Which one of the following is *not* a valid rule of Boolean algebra?

(a) $A + 1 = 1$ **(b)** $A = \overline{\overline{A}}$ **(c)** $AA = A$ **(d)** $A + 0 = A$

9. Which of the following rules states that if one input of an AND gate is always 1, the output is equal to the other input?

(a) $A + 1 = 1$ **(b)** $A + A = A$ **(c)** $A \cdot A = A$ **(d)** $A \cdot 1 = A$

10. According to DeMorgan's theorems, the following equality(s) are correct:

(a) $\overline{AB} = \overline{A} + \overline{B}$ **(b)** $\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$

(c) $\overline{A + B + C} = \overline{A}\,\overline{B}\,\overline{C}$ **(d)** all of these

11. The Boolean expression $X = AB + CD$ represents

(a) two ORs ANDed together **(b)** a 4-input AND gate

(c) two ANDs ORed together **(d)** an exclusive-OR

12. An example of a sum-of-products expression is

(a) $A + B(C + D)$ **(b)** $\overline{A}B + A\overline{C} + A\overline{B}C$

(c) $(\overline{A} + B + C)(A + \overline{B} + C)$ **(d)** both answers (a) and (b)

13. An example of a product-of-sums expression is

(a) $A(B + C) + A\overline{C}$ **(b)** $(A + B)(\overline{A} + B + \overline{C})$

(c) $\overline{A} + \overline{B} + BC$ **(d)** both answers (a) and (b)

14. An example of a standard SOP expression is

(a) $\overline{A}B + A\overline{B}C + AB\overline{D}$ **(b)** $A\overline{B}C + A\overline{C}D$

(c) $A\overline{B} + \overline{A}B + AB$ **(d)** $A\overline{B}C\overline{D} + \overline{A}B + \overline{A}$

15. A 3-variable Karnaugh map has

(a) eight cells **(b)** three cells **(c)** sixteen cells **(d)** four cells

16. In a 4-variable Karnaugh map, a 2-variable product term is produced by

(a) a 2-cell group of 1s **(b)** an 8-cell group of 1s

(c) a 4-cell group of 1s **(d)** a 4-cell group of 0s

17. On a Karnaugh map, grouping the 0s produces

(a) a product-of-sums expression **(b)** a sum-of-products expression

(c) a "don't care" condition **(d)** AND-OR logic

18. A 5-variable Karnaugh map has

(a) sixteen cells **(b)** thirty-two cells **(c)** sixty-four cells

PROBLEMS

SECTION 4–1 Boolean Operations and Expressions

1. Using Boolean notation, write an expression that is a 1 whenever one or more of its variables (A, B, C, and D) are 1s.

2. Write an expression that is a 1 only if all of its variables (A, B, C, D, and E) are 1s.
3. Write an expression that is a 1 when one or more of its variables (A, B, and C) are 0s.
4. Evaluate the following operations:
   - (a) $0 + 0 + 1$ (b) $1 + 1 + 1$ (c) $1 \cdot 0 \cdot 0$
   - (d) $1 \cdot 1 \cdot 1$ (e) $1 \cdot 0 \cdot 1$ (f) $1 \cdot 1 + 0 \cdot 1 \cdot 1$
5. Find the values of the variables that make each product term 1 and each sum term 0.
   - (a) AB (b) $A\overline{B}C$ (c) $A + B$ (d) $\overline{A} + B + \overline{C}$
   - (e) $\overline{A} + \overline{B} + C$ (f) $\overline{A} + B$ (g) $A\overline{B}\,\overline{C}$
6. Find the value of X for all possible values of the variables.
   - (a) $X = (A + B)C + B$ (b) $X = \overline{(A + B)}C$ (c) $X = A\overline{B}C + AB$
   - (d) $X = (A + B)(\overline{A} + B)$ (e) $X = (A + BC)(\overline{B} + \overline{C})$

SECTION 4–2 Laws and Rules of Boolean Algebra

7. Identify the law of Boolean algebra upon which each of the following equalities is based:
   - (a) $A\overline{B} + CD + A\overline{C}D + B = B + A\overline{B} + A\overline{C}D + CD$
   - (b) $AB\overline{C}D + \overline{ABC} = D\overline{C}BA + \overline{CBA}$
   - (c) $AB(CD + E\overline{F} + GH) = ABCD + ABE\overline{F} + ABGH$
8. Identify the Boolean rule(s) on which each of the following equalities is based:
   - (a) $\overline{\overline{AB + CD}} + \overline{EF} = AB + CD + \overline{EF}$ (b) $A\overline{A}B + AB\overline{C} + AB\overline{B} = AB\overline{C}$
   - (c) $A(BC + BC) + AC = A(BC) + AC$ (d) $AB(C + \overline{C}) + AC = AB + AC$
   - (e) $A\overline{B} + A\overline{B}C = A\overline{B}$ (f) $ABC + \overline{AB} + \overline{ABC}D = ABC + \overline{AB} + D$

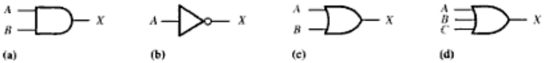
SECTION 4–3 DeMorgan's Theorems

9. Apply DeMorgan's theorems to each expression:
   - (a) $\overline{A + \overline{B}}$ (b) $\overline{\overline{A}B}$ (c) $\overline{A + B + C}$ (d) $\overline{ABC}$
   - (e) $\overline{A(B + C)}$ (f) $\overline{AB} + \overline{CD}$ (g) $\overline{AB + CD}$ (h) $\overline{(A + \overline{B})(\overline{C} + D)}$
10. Apply DeMorgan's theorems to each expression:
   - (a) $\overline{A\overline{B}(C + \overline{D})}$ (b) $\overline{AB(CD + EF)}$
   - (c) $\overline{(A + \overline{B} + C + \overline{D})} + \overline{ABC\overline{D}}$ (d) $\overline{\overline{(\overline{A} + B + C + D)}\,\overline{(AB\overline{C}\overline{D})}}$
   - (e) $\overline{\overline{AB}(CD + \overline{E}F)(\overline{AB} + \overline{CD})}$
11. Apply DeMorgan's theorems to the following:
   - (a) $\overline{\overline{\overline{(ABC)}\,\overline{(EFG)}} + \overline{\overline{(HIJ)}\,\overline{(KLM)}}}$ (b) $\overline{(A + \overline{BC} + CD)} + \overline{BC}$
   - (c) $\overline{\overline{\overline{(A + B)}\,\overline{(C + D)}\,\overline{(E + F)}\,\overline{(G + H)}}}$

SECTION 4–4 Boolean Analysis of Logic Circuits

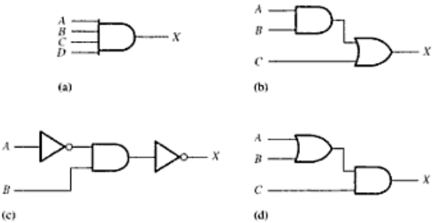
12. Write the Boolean expression for each of the logic gates in Figure 4–46.

FIGURE 4–46



13. Write the Boolean expression for each of the logic circuits in Figure 4-47.

FIGURE 4-47



14. Draw the logic circuit represented by each of the following expressions:

(a) $A + B + C$ **(b)** ABC **(c)** $AB + C$ **(d)** $AB + CD$

15. Draw the logic circuit represented by each expression:

(a) $A\overline{B} + \overline{A}B$ **(b)** $AB + \overline{A}\,\overline{B} + \overline{A}BC$

(c) $\overline{A}B(C + \overline{D})$ **(d)** $A + B[C + D(B + \overline{C})]$

16. Construct a truth table for each of the following Boolean expressions:

(a) $A + B$ **(b)** AB **(c)** $AB + BC$

(d) $(A + B)C$ **(e)** $(A + B)(\overline{B} + C)$

SECTION 4–5 Simplification Using Boolean Algebra

17. Using Boolean algebra techniques, simplify the following expressions as much as possible:

(a) $A(A + B)$ **(b)** $A(\overline{A} + AB)$ **(c)** $BC + \overline{B}C$

(d) $A(A + \overline{A}B)$ **(e)** $A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C$

18. Using Boolean algebra, simplify the following expressions:

(a) $(A + \overline{B})(A + C)$ **(b)** $\overline{A}B + \overline{A}B\overline{C} + \overline{A}BCD + \overline{A}B\overline{C}\,\overline{D}E$

(c) $AB + \overline{A}\,\overline{B}C + A$ **(d)** $(A + \overline{A})(AB + AB\overline{C})$

(e) $AB + (\overline{A} + \overline{B})C + AB$

19. Using Boolean algebra, simplify each expression:

(a) $BD + B(D + E) + \overline{D}(D + F)$ **(b)** $\overline{A}\,\overline{B}C + \overline{(A + B + \overline{C})} + \overline{A}\,\overline{B}\,\overline{C}D$

(c) $(B + BC)(B + \overline{B}C)(B + D)$ **(d)** $ABCD + AB(\overline{C}\,\overline{D}) + (\overline{A}\,\overline{B})CD$

(e) $ABC[AB + \overline{C}(BC + AC)]$

20. Determine which of the logic circuits in Figure 4–48 are equivalent.

SECTION 4–6 Standard Forms of Boolean Expressions

21. Convert the following expressions to sum-of-product (SOP) forms:

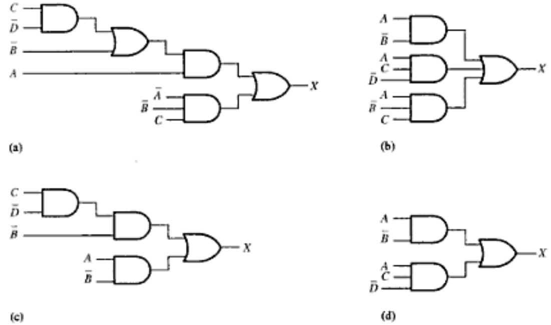
(a) $(A + B)(C + \overline{B})$ **(b)** $(A + \overline{B}C)C$ **(c)** $(A + C)(AB + AC)$

22. Convert the following expressions to sum-of-product (SOP) forms:

(a) $AB + CD(A\overline{B} + CD)$ **(b)** $AB(\overline{B}\,\overline{C} + BD)$ **(c)** $A + B[AC + (B + \overline{C})D]$

23. Define the domain of each SOP expression in Problem 21 and convert the expression to standard SOP form.

FIGURE 4–48



24. Convert each SOP expression in Problem 22 to standard SOP form.
25. Determine the binary value of each term in the standard SOP expressions from Problem 23.
26. Determine the binary value of each term in the standard SOP expressions from Problem 24.
27. Convert each standard SOP expression in Problem 23 to standard POS form.
28. Convert each standard SOP expression in Problem 24 to standard POS form.

SECTION 4–7 Boolean Expressions and Truth Tables

29. Develop a truth table for each of the following standard SOP expressions:
    (a) $A\overline{B}C + \overline{A}B\overline{C} + ABC$ (b) $\overline{X}\overline{Y}\overline{Z} + \overline{X}\overline{Y}Z + XY\overline{Z} + X\overline{Y}Z + \overline{X}YZ$
30. Develop a truth table for each of the following standard SOP expressions:
    (a) $\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D}$
    (b) $WXYZ + WXY\overline{Z} + \overline{W}XYZ + W\overline{X}YZ + WX\overline{Y}Z$
31. Develop a truth table for each of the SOP expressions:
    (a) $\overline{A}B + AB\overline{C} + \overline{A}\overline{C} + A\overline{B}C$ (b) $\overline{X} + Y\overline{Z} + WZ + X\overline{Y}Z$
32. Develop a truth table for each of the standard POS expressions:
    (a) $(\overline{A} + \overline{B} + \overline{C})(A + B + C)(A + \overline{B} + C)$
    (b) $(\overline{A} + B + \overline{C} + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)(\overline{A} + B + C + \overline{D})$
33. Develop a truth table for each of the standard POS expressions:
    (a) $(A + B)(A + C)(A + B + C)$
    (b) $(A + \overline{B})(A + \overline{B} + \overline{C})(B + C + \overline{D})(\overline{A} + B + \overline{C} + D)$
34. For each truth table in Figure 4–49, derive a standard SOP and a standard POS expression.

SECTION 4–8 The Karnaugh Map

35. Draw a 3-variable Karnaugh map and label each cell according to its binary value.
36. Draw a 4-variable Karnaugh map and label each cell according to its binary value.
37. Write the standard problem term for each cell in a 3-variable Karnaugh map.

SECTION 4–9 Karnaugh MAP SOP Minimization

38. Use a Karnaugh map to find the minimum SOP form for each expression:
    (a) $\overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + A\overline{B}C$ (b) $AC(\overline{B} + C)$
    (c) $\overline{A}(BC + B\overline{C}) + A(BC + B\overline{C})$ (d) $\overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + \overline{A}B\overline{C} + AB\overline{C}$

FIGURE 4–49

ABC	X
000	0
001	1
010	0
011	0
100	1
101	1
110	0
111	1

(a)

ABC	X
000	0
001	0
010	0
011	0
100	0
101	1
110	1
111	1

(b)

$ABCD$	X
0000	1
0001	1
0010	0
0011	1
0100	0
0101	1
0110	1
0111	0
1000	0
1001	1
1010	0
1011	0
1100	1
1101	0
1110	0
1111	0

(c)

$ABCD$	X
0000	0
0001	0
0010	1
0011	0
0100	1
0101	1
0110	0
0111	1
1000	0
1001	0
1010	0
1011	1
1100	1
1101	0
1110	0
1111	1

(d)

39. Use a Karnaugh map to simplify each expression to a minimum SOP form:

(a) $\overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + \overline{A}BC + AB\overline{C}$ **(b)** $AC[\overline{B} + B(B + \overline{C})]$

(c) $DE\overline{F} + \overline{D}E\overline{F} + \overline{D}\,\overline{E}\,\overline{F}$

40. Expand each expression to a standard SOP form:

(a) $AB + A\overline{B}C + ABC$ **(b)** $A + BC$

(c) $A\overline{B}\,\overline{C}D + AC\overline{D} + B\overline{C}D + \overline{A}BC\overline{D}$ **(d)** $A\overline{B} + A\overline{B}\,\overline{C}D + CD + B\overline{C}D + ABCD$

41. Minimize each expression in Problem 40 with a Karnaugh map.

42. Use a Karnaugh map to reduce each expression to a minimum SOP form:

(a) $A + B\overline{C} + CD$

(b) $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + ABCD + ABC\overline{D}$

(c) $\overline{A}B(\overline{C}\,\overline{D} + \overline{C}D) + AB(\overline{C}\,\overline{D} + \overline{C}D) + A\overline{B}\,\overline{C}D$

(d) $(\overline{A}\,\overline{B} + A\overline{B})(CD + C\overline{D})$

(e) $\overline{A}\,\overline{B} + A\overline{B} + \overline{C}\,\overline{D} + C\overline{D}$

43. Reduce the function specified in the truth table in Figure 4–50 to its minimum SOP form by using a Karnaugh map.

FIGURE 4–50

Inputs $A\ B\ C$	Output X
0 0 0	1
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	0
1 1 1	1

44. Use the Karnaugh map method to implement the minimum SOP expression for the logic function specified in the truth table in Figure 4–51.

► **FIGURE 4–51**

Inputs				Output
A	B	C	D	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

45. Solve Problem 44 for a situation in which the last six binary combinations are not allowed.

SECTION 4–10 Karnaugh Map POS Minimization

46. Use a Karnaugh map to find the minimum POS for each expression:
    - (a) $(A + B + C)(\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + C)$
    - (b) $(X + \overline{Y})(\overline{X} + Z)(X + \overline{Y} + \overline{Z})(\overline{X} + \overline{Y} + Z)$
    - (c) $A(B + \overline{C})(\overline{A} + C)(A + \overline{B} + C)(\overline{A} + B + \overline{C})$
47. Use a Karnaugh map to simplify each expression to minimum POS form:
    - (a) $(A + \overline{B} + C + \overline{D})(\overline{A} + B + \overline{C} + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D})$
    - (b) $(X + \overline{Y})(W + \overline{Z})(\overline{X} + \overline{Y} + \overline{Z})(W + X + Y + Z)$
48. For the function specified in the truth table of Figure 4–50, determine the minimum POS expression using a Karnaugh map.
49. Determine the minimum POS expression for the function in the truth table of Figure 4–51.
50. Convert each of the following POS expressions to minimum SOP expressions using a Karnaugh map:
    - (a) $(A + \overline{B})(A + \overline{C})(\overline{A} + \overline{B} + C)$
    - (b) $(\overline{A} + B)(\overline{A} + \overline{B} + \overline{C})(B + \overline{C} + D)(A + \overline{B} + C + \overline{D})$

SECTION 4–11 Five-Variable Karnaugh Maps

51. Minimize the following SOP expression using a Karnaugh map:

$$\begin{aligned} X = {} & \overline{A}B\overline{C}D\overline{E} + \overline{A}\,\overline{B}\,\overline{C}DE + A\overline{B}\,\overline{C}DE + AB\overline{C}\,\overline{D}\,\overline{E} + \overline{A}BCD\overline{E} + \overline{A}BC\overline{D}E \\ & + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} + \overline{A}\,\overline{B}CDE + AB\overline{C}D\overline{E} + AB\overline{C}DE \end{aligned}$$

52. Apply the Karnaugh map method to minimize the following SOP expression:

$$\begin{aligned} A = {} & \overline{V}WXYZ + V\overline{W}XYZ + VW\overline{X}YZ + VWX\overline{Y}Z + VWXY\overline{Z} + \overline{V}\,\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} \\ & + \overline{V}\,\overline{W}\,\overline{X}Y\overline{Z} + \overline{V}\,\overline{W}X\overline{Y}\,\overline{Z} + \overline{V}W\overline{X}\,\overline{Y}\,\overline{Z} \end{aligned}$$

ANSWERS

SECTION REVIEWS

SECTION 4–1 Boolean Operations and Expressions

1. $\overline{A} = \overline{0} = 1$ 2. $A = 1, B = 1, C = 0; \overline{A} + \overline{B} + C = \overline{1} + \overline{1} + 0 = 0 + 0 + 0 = 0$

3. $A = 1, B = 0, C = 1; A\overline{B}C = 1 \cdot \overline{0} \cdot 1 = 1 \cdot 1 \cdot 1 = 1$

SECTION 4–2 Laws and Rules of Boolean Algebra

1. $A + (B + C + D) = (A + B + C) + D$ 2. $A(B + C + D) = AB + AC + AD$

SECTION 4–3 DeMorgan's Theorems

1. (a) $\overline{ABC} + \overline{(\overline{D} + E)} = \overline{A} + \overline{B} + \overline{C} + D\overline{E}$ (b) $\overline{(A + B)C} = \overline{A}\,\overline{B} + \overline{C}$

(c) $\overline{A + B + C} + \overline{\overline{D}E} = \overline{A}\,\overline{B}\,\overline{C} + D + \overline{E}$

SECTION 4–4 Boolean Analysis of Logic Circuits

1. $(C + D)B + A$

2. Abbreviated truth table: The expression is a 1 when A is 1 or when B and C are 1s or when B and D are 1s. The expression is 0 for all other variable combinations.

SECTION 4–5 Simplification Using Boolean Algebra

1. (a) $A + AB + A\overline{B}C = A$ (b) $(\overline{A} + B)C + ABC = C(\overline{A} + B)$

(c) $A\overline{B}C(BD + CDE) + A\overline{C} = A(\overline{C} + \overline{B}DE)$

2. (a) *Original*: 2 AND gates, 1 OR gate, 1 inverter; *Simplified*: No gates (straight connection)

(b) *Original*: 2 OR gates, 2 AND gates, 1 inverter; *Simplified*: 1 OR gate, 1 AND gate, 1 inverter

(c) *Original*: 5 AND gates, 2 OR gates, 2 inverters; *Simplified*: 2 AND gates, 1 OR gate, 2 inverters

SECTION 4–6 Standard Forms of Boolean Expressions

1. (a) SOP (b) standard POS (c) standard SOP (d) POS

2. (a) $AB\overline{C}\,\overline{D} + AB\overline{C}D + ABC\overline{D} + ABCD + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}B\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$

(c) Already standard

3. (b) Already standard

(d) $(A + \overline{B} + \overline{C})(A + \overline{B} + C)(A + B + \overline{C})(A + B + C)$

SECTION 4–7 Boolean Expressions and Truth Tables

1. $2^5 = 32$ 2. $0110 \longrightarrow \overline{W}XY\overline{Z}$ 3. $1100 \longrightarrow \overline{W} + \overline{X} + Y + Z$

SECTION 4–8 The Karnaugh Map

1. (a) upper left cell: 000 (b) lower right cell: 101 (c) lower left cell: 100

(d) upper right cell: 001

2. (a) upper left cell: $\overline{X}\,\overline{Y}\,\overline{Z}$ (b) lower right cell: $X\overline{Y}Z$ (c) lower left cell: $X\overline{Y}\,\overline{Z}$

(d) upper right cell: $\overline{X}\,\overline{Y}Z$

3. (a) upper left cell: 0000 (b) lower right cell: 1010 (c) lower left cell: 1000

(d) upper right cell: 0010

4. (a) upper left cell: $\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$ (b) lower right cell: $W\overline{X}Y\overline{Z}$ (c) lower left cell: $W\overline{X}\,\overline{Y}\,\overline{Z}$

(d) upper right cell: $\overline{W}\,\overline{X}Y\overline{Z}$

SECTION 4–9 Karnaugh Map SOP Minimization

1. 8-cell map for 3 variables; 16-cell map for 4 variables
2. $AB + B\bar{C} + \bar{A}\bar{B}C$
3. (a) $\bar{A}\bar{B}\bar{C} + \bar{A}BC + ABC + AB\bar{C}$
   (b) $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C} + A\bar{B}C$
   (c) $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$
   (d) $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + AB\bar{C}D + A\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D}$

SECTION 4–10 Karnaugh Map POS Minimization

1. In mapping a POS expression, 0s are placed in cells whose value makes the standard sum term zero; and in mapping an SOP expression 1s are placed in cells having the same values as the product terms.
2. 0 in the 1011 cell: $\bar{A} + B + \bar{C} + \bar{D}$ 3. 1 in the 0010 cell: $\bar{A}\bar{B}C\bar{D}$

SECTION 4–11 Five-Variable Karnaugh Maps

1. There are 32 combinations of 5 variables ($2^5 = 32$).
2. $X = 1$ because the function is 1 for all possible combinations of 5 variables.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

4–1 $\bar{A} + B = 0$ when $A = 1$ and $B = 0$.

4–2 $\bar{A}\bar{B} = 1$ when $A = 0$ and $B = 0$. **4–3** XYZ

4–4 $W + X + Y + Z$ **4–5** $ABC\bar{D}\bar{E}$ **4–6** $(A + \bar{B} + \bar{C}D)\bar{E}$

4–7 $\overline{ABCD} = \bar{A} + \bar{B} + \bar{C} + \bar{D}$ **4–8** $A\bar{B}$ **4–9** CD

4–10 $AB\bar{C} + \bar{A}C + \bar{A}\bar{B}$

4–11 $\bar{A} + \bar{B} + \bar{C}$ **4–12** $\bar{A}B\bar{C} + AB + A\bar{C} + A\bar{B} + \bar{B}\bar{C}$

4–13 $W\bar{X}YZ + W\bar{X}Y\bar{Z} + W\bar{X}Y\bar{Z} + \bar{W}\bar{X}Y\bar{Z} + WX\bar{Y}Z + WX\bar{Y}\bar{Z}$

4–14 011, 101, 110, 010, 111. Yes

4–15 $(A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + B + C)(\bar{A} + B + C)$

4–16 010, 100, 001, 111, 011. Yes **4–17** SOP and POS expressions are equivalent.

4–18 See Table 4–9. **4–19** See Table 4–10.

▼ TABLE 4–9

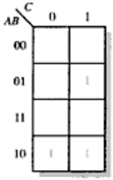
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

▼ TABLE 4–10

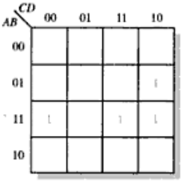
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

4–20 The SOP and POS expressions are equivalent. **4–21** See Figure 4–52.

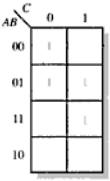
4–22 See Figure 4–53. **4–23** See Figure 4–54. **4–24** See Figure 4–55.


C
AB
0
1
00
01
11
10
1
1
1

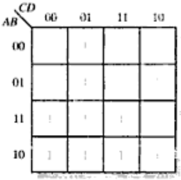
▲ **FIGURE 4–52**


CD
AB
00
01
11
10
00
01
11
10
1
1
1
1

▲ **FIGURE 4–53**


C
AB
0
1
00
01
11
10
1
1
1
1

▲ **FIGURE 4–54**


CD
AB
00
01
11
10
00
01
11
10

▲ **FIGURE 4–55**

4–25 No other ways **4–26** $X = B + \overline{A}C + A\overline{C}D + C\overline{D}$

4–27 $X = \overline{D} + A\overline{B}C + B\overline{C} + \overline{A}B$

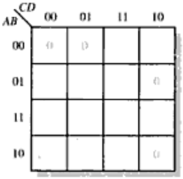
4–28 $Q = X + Y$ **4–29** $Q = \overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}Z + \overline{W}YZ$ **4–30** See Figure 4–56.

4–31 $Q = (X + \overline{Y})(X + \overline{Z})(\overline{X} + Y + Z)$

4–32 $Q = (\overline{X} + \overline{Y} + Z)(\overline{W} + \overline{X} + Z)(W + X + Y + Z)(W + \overline{X} + Y + \overline{Z})$

4–33 $Q = \overline{Y}\,\overline{Z} + \overline{X}\,\overline{Z} + \overline{W}Y + \overline{X}\,\overline{Y}Z$

4–34 $Y = \overline{D}\,\overline{E} + \overline{A}\,\overline{E} + \overline{B}\,\overline{C}\,\overline{E}$


CD
AB
00
01
11
10
00
01
11
10
0
0
0
0

▲ **FIGURE 4–56**

SELF-TEST

1. (d) **2.** (a) **3.** (b) **4.** (c) **5.** (d) **6.** (b) **7.** (a) **8.** (b)
9. (d) **10.** (d) **11.** (c) **12.** (b) **13.** (b) **14.** (c) **15.** (a) **16.** (c)
17. (a) **18.** (b)

5

COMBINATIONAL LOGIC

CHAPTER OBJECTIVES

- Analyze basic combinational logic circuits, such as AND-OR, AND-OR-Invert, exclusive-OR, exclusive-NOR, and other general combinational networks
- Use AND-OR and AND-OR-Invert circuits to implement sum-of-products (SOP) and product-of-sums (POS) expressions
- Write the Boolean output expression for any combinational logic circuit
- Develop a truth table from the output expression for a combinational logic circuit
- Use the Karnaugh map to expand an output expression containing terms with missing variables into a full SOP form
- Design a combinational logic circuit for a given Boolean output expression
- Design a combinational logic circuit for a given truth table
- Simplify a combinational logic circuit to its minimum form
- Use NAND gates to implement any combinational logic function
- Use NOR gates to implement any combinational logic function

INTRODUCTION

In Chapters 3 and 4, logic gates were discussed on an individual basis and in simple combinations. You were introduced to SOP and POS implementations, which are basic forms of combinational logic. When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is in the category of **combinational logic**. In combinational logic, the output level is at all times dependent on the combination of input levels. This chapter expands on the material introduced in earlier chapters with a coverage of the analysis, design, and troubleshooting of various combinational logic circuits.

5–1 BASIC COMBINATIONAL LOGIC CIRCUITS

In Chapter 4, you learned that SOP expressions are implemented with an AND gate for each product term and one OR gate for summing all of the product terms. This SOP implementation is called AND-OR logic and is the basic form for realizing standard Boolean functions. In this section, the AND-OR and the AND-OR-Invert are examined; and the exclusive-OR and exclusive-NOR gates, which are actually a form of AND-OR logic, are also covered.

After completing this section, you should be able to

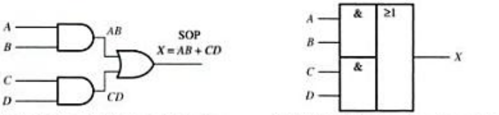
- Analyze and apply AND-OR circuits
- Analyze and apply AND-OR-Invert circuits
- Analyze and apply exclusive-OR gates
- Analyze and apply exclusive-NOR gates

AND-OR Logic

Figure 5–1(a) shows an AND-OR circuit consisting of two 2-input AND gates and one 2-input OR gate; Figure 5–1(b) is the ANSI standard rectangular outline symbol. The Boolean expressions for the AND gate outputs and the resulting SOP expression for the output X are shown on the diagram. In general, and AND-OR circuit can have any number of AND gates each with any number of inputs.

The truth table for a 4-input AND-OR logic circuit is shown in Table 5–1. The intermediate AND gate outputs (the AB and CD columns) are also shown in the table.

▶ FIGURE 5–1



(a) Logic diagram (ANSI standard distinctive shape symbols)

(b) ANSI standard rectangular outline symbol

▶ TABLE 5–1

INPUTS						OUTPUT
A	**B**	**C**	**D**	**AB**	**CD**	**X**
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

An AND-OR circuit directly implements an SOP expression, assuming the complements (if any) of the variables are available. The operation of the AND-OR circuit in Figure 5–1 is stated as follows:

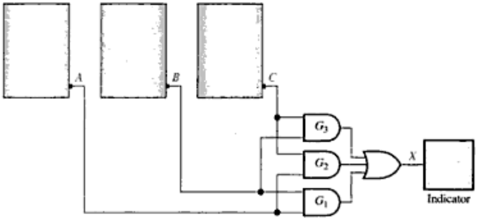
For a 4-input AND-OR logic circuit, the output X is HIGH (1) if both input A and input B are HIGH (1) or both input C and input D are HIGH (1).

EXAMPLE 5–1

In a certain chemical-processing plant, a liquid chemical is used in a manufacturing process. The chemical is stored in three different tanks. A level sensor in each tank produces a HIGH voltage when the level of chemical in the tank drops below a specified point.

Design a circuit that monitors the chemical level in each tank and indicates when the level in any two of the tanks drops below the specified point.

Solution The AND-OR circuit in Figure 5–2 has inputs from the sensors on tanks *A*, *B*, and *C* as shown. The AND gate G_1 checks the levels in tanks *A* and *B*, gate G_2 checks tanks *A* and *C*, and gate G_3 checks tanks *B* and *C*. When the chemical level in any two of the tanks gets too low, one of the AND gates will have HIGHs on both of its inputs causing its output to be HIGH, and so the final output *X* from the OR gate is HIGH. This HIGH input is then used to activate an indicator such as a lamp or audible alarm, as shown in the figure.



▲ **FIGURE 5–2**

Supplementary Problem Write the Boolean SOP expression for the AND-OR logic in Figure 5–2.

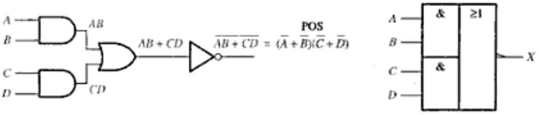
AND-OR-Invert Logic

When the output of an AND-OR circuit is complemented (inverted), it results in an AND-OR-Invert circuit. Recall that AND-OR logic directly implements SOP expressions. POS expressions can be implemented with AND-OR-Invert logic. This is illustrated as follows, starting with a POS expression and developing the corresponding AND-OR-Invert expression.

$$X = (\overline{A} + \overline{B})(\overline{C} + \overline{D}) = (\overline{AB})(\overline{CD}) = \overline{\overline{(\overline{AB})(\overline{CD})}} = \overline{\overline{\overline{AB}} + \overline{\overline{\overline{CD}}}} = \overline{AB + CD}$$

The logic diagram in Figure 5–3(a) shows an AND-OR-Invert circuit and the development of the POS output expression. The ANSI standard rectangular outline symbol is shown in

▶ **FIGURE 5–3**


A
B
C
D
AB
CD
AB + CD
POS
$\overline{AB + CD} = (\overline{A} + \overline{B})(\overline{C} + \overline{D})$
A
B
C
D
&
≥1
&
X

(a) (b)

part (b). In general, an AND-OR-Invert circuit can have any number of AND gates each with any number of inputs.

The operation of the AND-OR-Invert circuit in Figure 5–3 is stated as follows:

For a 4-input AND-OR-Invert logic circuit, the output X is LOW (0) if both input A and input B are HIGH (1) or both input C and input D are HIGH (1).

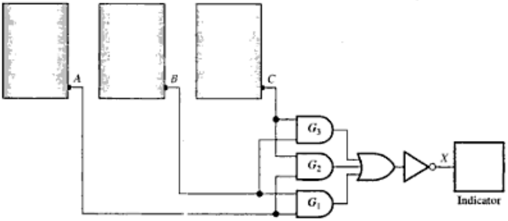
A truth table can be developed from the AND-OR truth table in Table 5–1 by simply changing all 1s to 0s and all 0s to 1s in the output column.

EXAMPLE 5–2

The sensors in the chemical tanks of Example 5–1 are being replaced by a new model that produces a LOW voltage instead of a HIGH voltage when the level of the chemical in the tank drops below a critical point.

Modify the circuit in Figure 5–2 to operate with the different input levels and still produce a HIGH output to activate the indicator when the level in any two of the tanks drops below the critical point. Show the logic diagram.

Solution The AND-OR-Invert circuit in Figure 5–4 has inputs from the sensors on tanks A, B, and C as shown. The AND gate G_1 checks the levels in tanks A and B, gate G_2 checks tanks A and C, and gate G_3 checks tanks B and C. When the chemical level in any two of the tanks gets too low, each AND gate will have a LOW on at least one input causing its output to be LOW and, thus, the final output X from the inverter is HIGH. This HIGH output is then used to activate an indicator.

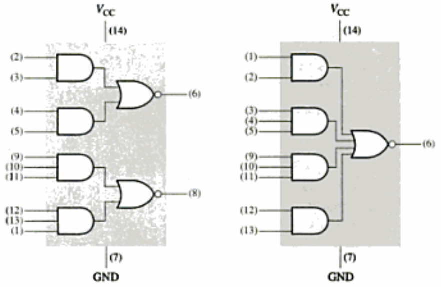


▲ FIGURE 5–4

Supplementary Problem Write the Boolean expression for the AND-OR-Invert logic in Figure 5–4 and show that the output is HIGH (1) when any two of the inputs A, B, and C are LOW (0).

AND-OR-INVERT INTEGRATED CIRCUITS

The 74LS51 and the 74LS54 are examples of AND-OR-Invert logic. The 74LS51 includes two separate AND-OR-Invert circuits in a single package. One circuit has two 2-input AND gates and the other circuit has two 3-input AND gates, as shown in Figure 5–5(a). The 74LS54 is a single AND-OR-Invert circuit that has two 2-input AND gates and two 3-input AND gates, as shown in Figure 5–5(b). Notice that the inversion is indicated by a bubble on the output of the OR gates in each case, showing that the OR-Invert part of the circuit is effectively a NOR gate.

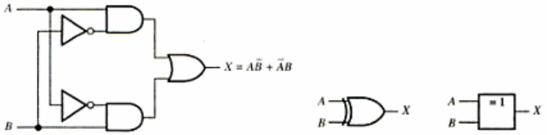


(a) 74LS51 (b) 74LS54

▲ **FIGURE 5–5**

Exclusive-OR Logic

The exclusive-OR gate was introduced in Chapter 3. Although, because of its importance, this circuit is considered a type of logic gate with its own unique symbol, it is actually a combination of two AND gates, one OR gate, and two inverters, as shown in Figure 5–6(a). The two ANSI standard logic symbols are shown in parts (b) and (c).



(a) Logic diagram (b) ANSI distinctive shape symbol (c) ANSI rectangular outline symbol

▲ **FIGURE 5–6**

The output expression for the circuit in Figure 5–6 is

$$X = A\bar{B} + \bar{A}B$$

Evaluation of this expression results in the truth table in Table 5–2. Notice that the output is HIGH only when the two inputs are at opposite levels. A special exclusive-OR operator $\oplus$ is often used, so the expression $X = A\bar{B} + \bar{A}B$ can be stated as "X is equal to A exclusive-OR B" and can be written as

$$X = A \oplus B$$

TABLE 5–2

Truth table for exclusive-OR

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR Logic

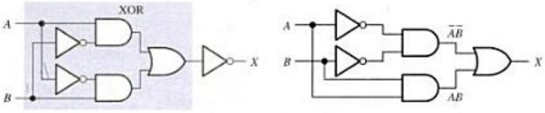
As you know, the complement of the exclusive-OR function is the exclusive-NOR, which is derived as follows:

$$X = \overline{A\overline{B} + \overline{A}B} = \overline{(A\overline{B})}\,\overline{(\overline{A}B)} = (\overline{A} + B)(A + \overline{B}) = \overline{A}\,\overline{B} + AB$$

Notice that the output X is HIGH only when the two inputs, A and B, are at the same level.

The exclusive-NOR can be implemented by simply inverting the output of an exclusive-OR, as shown in Figure 5–7(a), or by directly implementing the expression $\overline{A}\,\overline{B} + AB$, as shown in part (b).

FIGURE 5–7



(a) $X = \overline{A\overline{B} + \overline{A}B}$ (b) $X = \overline{A}\overline{B} + AB$

SECTION 5–1 REVIEW

Answers are at the end of the chapter.

1. Determine the output (1 or 0) of a 4-variable AND-OR-Invert circuit for each of the following input conditions:
   (a) $A = 1, B = 0, C = 1, D = 0$ (b) $A = 1, B = 1, C = 0, D = 1$
   (c) $A = 0, B = 1, C = 1, D = 1$.
2. Determine the output (1 or 0) of an exclusive-OR gate for each of the following input conditions:
   (a) $A = 1, B = 0$ (b) $A = 1, B = 1$
   (c) $A = 0, B = 1$ (d) $A = 0, B = 0$.
3. Develop the truth table for a certain 3-input logic circuit with the output expression $X = A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + ABC$.
4. Draw the logic diagram for an exclusive-NOR circuit.

5–2 IMPLEMENTING COMBINATIONAL LOGIC

In this section, examples are used to illustrate how to implement a logic circuit from a Boolean expression or a truth table. Minimization of a logic circuit using the methods covered in Chapter 4 is also discussed.

After completing this section, you should be able to

- Implement a logic circuit from a Boolean expression
- Implement a logic circuit from a truth table
- Minimize a logic circuit

From a Boolean Expression to a Logic Circuit

Let us examine the following Boolean expression:

$$X = AB + CDE$$

A brief inspection shows that this expression is composed of two terms, AB and CDE, with a domain of five variables. The first term is formed by ANDing A with B, and the second term is formed by ANDing C, D, and E. The two terms are then ORed to form the output X. These operations are indicated in the structure of the expression as follows:

AND

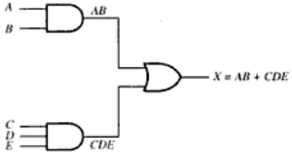
$$X = AB + CDE$$

OR

Note that in this particular expression, the AND operations forming the two individual terms, AB and CDE, must be performed *before* the terms can be ORed.

To implement this Boolean expression, a 2-input AND gate is required to form the term AB, and a 3-input AND gate is needed to form the term CDE. A 2-input OR gate is then required to combine the two AND terms. The resulting logic circuit is shown in Figure 5–8.

FIGURE 5–8



As another example, let us implement the following expression:

$$X = AB(C\overline{D} + EF)$$

A breakdown of this expression shows that the terms AB, and $C\overline{D} + EF$ are ANDed. The term $C\overline{D} + EF$ is formed by first ANDing C and $\overline{D}$ and ANDing E and F, and then ORing these two terms. This structure is indicated in relation to the expression as follows:

AND

NOT

OR

$$X = AB(C\overline{D} + EF)$$

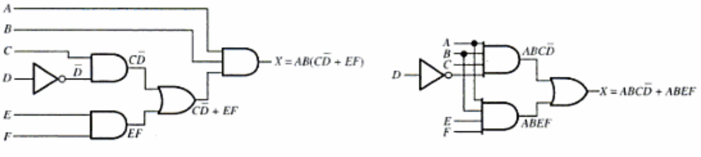
AND

Before the expression can be formed, you must have the term $C\overline{D} + EF$; but before you can get this term, you must have the terms $C\overline{D}$ and EF; but before you can get the term $C\overline{D}$, you must have $\overline{D}$. So, as you can see, the logic operations must be done in the proper order.

The logic gates required to implement $X = AB(C\overline{D} + EF)$ are as follows:

1. One inverter to form $\overline{D}$
2. Two 2-input AND gates to form $C\overline{D}$ and EF
3. One 2-input OR gate to form $C\overline{D} + EF$
4. One 3-input AND gate to form X

The logic circuit for this expression is shown in Figure 5–9(a). Notice that there is a maximum of four gates and an inverter between an input and output in this circuit (from input D to output). Often, the total propagation delay time through a logic circuit is a major consideration. Propagation delays are additive, so the more gates or inverters between input and output, the greater the propagation delay time.



(a)

(b) Sum-of-products implementation of the circuit in part (a)

▲ **FIGURE 5–9**

Unless an intermediate term, such as $C\overline{D} + EF$ in Figure 5–9(a), is required as an output for some other purpose, it is usually best to reduce a circuit to its SOP form. The expression is converted to SOP as follows, and the resulting circuit is shown in Figure 5–9(b).

$$AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$$

From a Truth Table to a Logic Circuit

If you begin with a truth table instead of an expression, you can write the SOP expression from the truth table and then implement the logic circuit. Table 5–3 specifies a logic function.

▶ **TABLE 5–3**

INPUTS			OUTPUT	
A	**B**	**C**	**X**	**PRODUCT TERM**
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

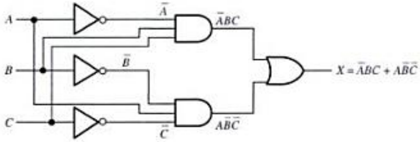
The Boolean SOP expression obtained from the truth table by ORing the product terms for which $X = 1$ is

$$X = \overline{A}BC + A\overline{B}\,\overline{C}.$$

The first term in the expression is formed by ANDing the three variables $\overline{A}$, B, and C. The second term is formed by ANDing the three variables A, $\overline{B}$, and $\overline{C}$.

The logic gates required to implement this expression are as follows: three inverters to form the $\overline{A}$, $\overline{B}$, and $\overline{C}$ variables; two 3-input AND gates to form the terms $\overline{A}BC$ and $A\overline{B}\,\overline{C}$; and one 2-input OR gate to form the final output function, $\overline{A}BC + A\overline{B}\,\overline{C}$.

The implementation of this logic function is illustrated in Figure 5–10.


A
B
C
$\bar{A}$
$\bar{B}$
$\bar{C}$
$\bar{A}BC$
$A\bar{B}\bar{C}$
$X = \bar{A}BC + A\bar{B}\bar{C}$

▲ **FIGURE 5–10**

EXAMPLE 5–3

Design a logic circuit to implement the operation specified in the truth table of Table 5–4.

▼ **TABLE 5–4**

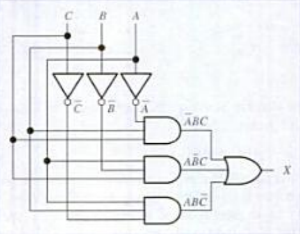
INPUTS			OUTPUT	
A	B	C	X	PRODUCT TERM
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{A}BC$
1	0	0	0	
1	0	1	1	$A\bar{B}C$
1	1	0	1	$AB\bar{C}$
1	1	1	0	

Solution Notice that $X = 1$ for only three of the input conditions. Therefore, the logic expression is

$$X = \bar{A}BC + A\bar{B}C + AB\bar{C}.$$

The logic gates required are three inverters, three 3-input AND gates and one 3-input OR gate. The logic circuit is shown in Figure 5–11.

► **FIGURE 5–11**


C
B
A
$\bar{C}$
$\bar{B}$
$\bar{A}$
$\bar{A}BC$
$A\bar{B}C$
$AB\bar{C}$
X

Supplementary Problem Determine if the logic circuit of Figure 5–11 can be simplified.

EXAMPLE 5–4

Develop a logic circuit with four input variables that will only produce a 1 output when exactly three input variables are 1s.

Solution Out of sixteen possible combinations of four variables, the combinations in which there are exactly three 1s are listed in Table 5–5, along with the corresponding product term for each.

TABLE 5–5

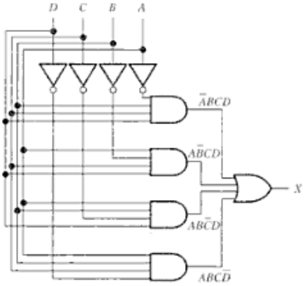
A	B	C	D	PRODUCT TERM
0	1	1	1	$\overline{A}BCD$
1	0	1	1	$ABCD$
1	1	0	1	$AB\overline{C}D$
1	1	1	0	$ABC\overline{D}$

The product terms are ORed to get the following expression:

$$X = \overline{A}BCD + A\overline{B}CD + AB\overline{C}D + ABC\overline{D}$$

This expression is implemented in Figure 5–12 with AND-OR logic.

FIGURE 5–12

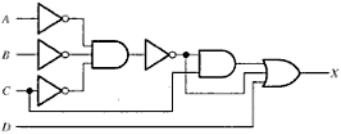


Supplementary Problem Determine if the logic circuit of Figure 5–12 can be simplified.

EXAMPLE 5–5

Reduce the combinational logic circuit in Figure 5–13 to a minimum form.

▶ **FIGURE 5–13**



Solution The expression for the output of the circuit is

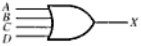
$$X = (\overline{\overline{A}\,\overline{B}\,\overline{C}})C + \overline{\overline{A}\,\overline{B}\,\overline{C}} + D$$

Applying DeMorgan's theorem and Boolean algebra,

$$\begin{aligned} X &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}})C + \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + D \\ &= AC + BC + CC + A + B + C + D \\ &= AC + BC + C + A + B + \cancel{C} + D \\ &= C(A + B + 1) + A + B + D \\ X &= A + B + C + D \end{aligned}$$

The simplified circuit is a 4-input OR gate as shown in Figure 5–14.

▶ **FIGURE 5–14**

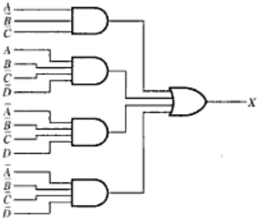


Supplementary Problem Verify the minimized expression $A + B + C + D$ using a Karnaugh map.

EXAMPLE 5–6

Minimize the combinational logic circuit in Figure 5–15. Inverters for the complemented variables are not shown.

▶ **FIGURE 5–15**


A
$\overline{B}$
$\overline{C}$
A
$\overline{B}$
$\overline{C}$
$\overline{D}$
$\overline{A}$
$\overline{B}$
$\overline{C}$
D
$\overline{A}$
$\overline{B}$
$\overline{C}$
$\overline{D}$
X

Solution The output expression is

$$X = A\overline{B}\,\overline{C} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

Expanding the first term to include the missing variables D and $\overline{D}$,

$$\begin{aligned} X &= A\overline{B}\,\overline{C}(D + \overline{D}) + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \\ &= A\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \end{aligned}$$

This expanded SOP expression is mapped and simplified on the Karnaugh map in Figure 5–16(a). The simplified implementation is shown in part (b). Inverters are not shown.

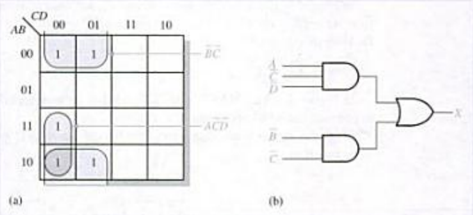


FIGURE 5–16

Supplementary Problem Develop the POS equivalent of the circuit in Figure 5–16(b).

SECTION 5–2 REVIEW

1. Implement the following Boolean expressions as they are stated:
   (a) $X = ABC + AB + AC$ (b) $X = AB(C + DE)$
2. Develop a logic circuit that will produce a 1 on its output only when all three inputs are 1s or when all three inputs are 0s.
3. Reduce the circuits in Question 1 to minimum SOP form.

5–3 THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

Up to this point, you have studied combinational circuits implemented with AND gates, OR gates, and inverters. In this section, the universal property of the NAND gate and the NOR gate is discussed. The universality of the NAND gate means that it can be used as an inverter and that combinations of NAND gates can be used to implement the AND, OR, and NOR operations. Similarly, the NOR gate can be used to implement the inverter, AND, OR, and NAND operations.

After completing this section, you should be able to

- Use NAND gates to implement the inverter, the AND gate, the OR gate, and the NOR gate
- Use NOR gates to implement the inverter, the AND gate, the OR gate, and the NAND gate

The NAND Gate as a Universal Logic Element

The NAND gate is a **universal gate** because it can be used to produce the NOT, the AND, the OR, and the NOR functions. An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single input, as shown in Figure 5–17(a) for a 2-input gate. An AND function can be generated by the use of NAND gates alone, as shown in Figure 5–17(b). An OR function can be produced with only NAND gates, as illustrated in part (c). Finally, a NOR function is produced as shown in part (d).

In Figure 5–17(b), a NAND gate is used to invert (complement) a NAND output to form the AND function, as indicated in the following equation:

$$X = \overline{\overline{AB}} = AB$$

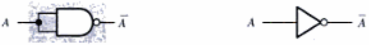
In Figure 5–17(c), NAND gates G_1 and G_2 are used to invert the two input variables before they are applied to NAND gate G_3. The final OR output is derived as follows by application of DeMorgan's theorem:

$$X = \overline{\overline{A}\,\overline{B}} = A + B$$

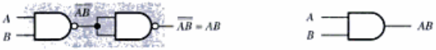
In Figure 5–17(d), NAND gate G_4 is used as an inverter connected to the circuit of part (c) to produce the NOR operation $\overline{A + B}$.

From the above discussion we conclude that the NAND gates can be used to produce any logic function.

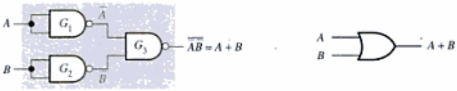
▶ **FIGURE 5–17**


A
$\overline{A}$
A
$\overline{A}$

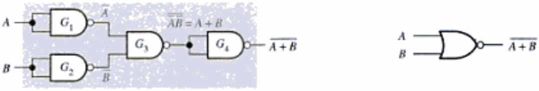
(a) A NAND gate used as an inverter


A
B
$\overline{AB}$
$\overline{\overline{AB}} = AB$
A
B
AB

(b) Two NAND gates used as an AND gate


A
B
G_1
G_2
G_3
$\overline{A}$
$\overline{B}$
$\overline{\overline{A}\,\overline{B}} = A + B$
A
B
A + B

(c) Three NAND gates used as an OR gate


A
B
G_1
G_2
G_3
G_4
$\overline{A}$
$\overline{B}$
$\overline{\overline{A}\,\overline{B}} = A + B$
$\overline{A + B}$
A
B
$\overline{A + B}$

(d) Four NAND gates used as a NOR gate

The NOR Gate as a Universal Logic Element

Like the NAND gate, the NOR gate can be used to produce the NOT, AND, OR, and NAND functions. A NOT circuit, or inverter, can be made from a NOR gate by connecting all of the inputs together to effectively create a single input, as shown in Figure 5–18(a) with a 2-input example. Also, an OR gate can be produced from NOR gates, as illustrated in Figure 5–18(b).

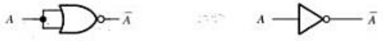
An AND gate can be constructed by the use of NOR gates, as shown in Figure 5–18(c). In this case the NOR gates G_1 and G_2 are used as inverters, and the final output is derived by the use of DeMorgan's theorem as follows:

$$X = \overline{\overline{A} + \overline{B}} = AB$$

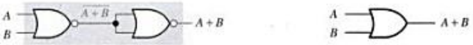
Figure 5–18(d) shows how NOR gates are used to form a NAND function.

Thus, NOR gates, similar to NAND gates, can be used to produce any logic function.

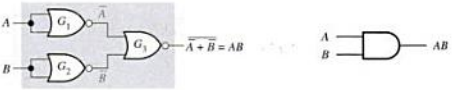
▶ FIGURE 5–18


A
$\overline{A}$
A
$\overline{A}$

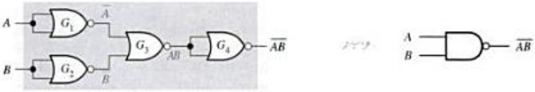
(a) A NOR gate used as an inverter


A
B
$\overline{A+B}$
A + B
A
B
A + B

(b) Two NOR gates used as an OR gate


A
B
G_1
G_2
G_3
$\overline{A}$
$\overline{B}$
$\overline{\overline{A}+\overline{B}} = AB$
A
B
AB

(c) Three NOR gates used as an AND gate


A
B
G_1
G_2
G_3
G_4
$\overline{A}$
$\overline{B}$
AB
$\overline{AB}$
A
B
$\overline{AB}$

(d) Four NOR gates used as a NAND gate

SECTION 5–3 REVIEW

1. Use NAND gates to implement each expression:
   (a) $X = \overline{A} + B$ (b) $X = A\overline{B}$
2. Use NOR gates to implement each expression:
   (a) $X = \overline{A} + B$ (b) $X = A\overline{B}$

5–4 COMBINATIONAL LOGIC USING NAND AND NOR GATES

In this section, you will see how NAND and NOR gates can be used to implement a logic function. Recall from Chapter 3 that the NAND gate also exhibits an equivalent operation called the negative-OR and that the NOR gate exhibits an equivalent operation called the negative-AND. You will see how the use of the appropriate symbols to represent the equivalent operations makes "reading" a logic diagram easier.

After completing this section, you should be able to

- Use NAND gates to implement a logic function
- Use NOR gates to implement a logic function
- Use the appropriate dual symbol in a logic diagram

NAND Logic

As you have learned, a NAND gate can function as either a NAND or a negative-OR because, by DeMorgan's theorem,

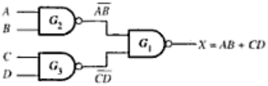
$$\underset{\text{NAND}}{\overline{AB}} = \underset{\text{negative-OR}}{\overline{A} + \overline{B}}$$

Consider the NAND logic in Figure 5–19. The output expression is developed in the following steps:

$$\begin{aligned} X &= \overline{(\overline{AB})(\overline{CD})} \\ &= \overline{(\overline{A} + \overline{B})(\overline{C} + \overline{D})} \\ &= \overline{(\overline{A} + \overline{B})} + \overline{(\overline{C} + \overline{D})} \\ &= \overline{\overline{A}}\,\overline{\overline{B}} + \overline{\overline{C}}\,\overline{\overline{D}} \\ &= AB + CD \end{aligned}$$

FIGURE 5–19

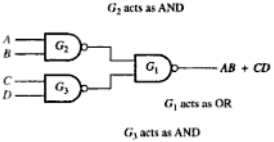
NAND logic for $X = AB + CD$.


A
B
G2
$\overline{AB}$
G1
X = AB + CD
C
D
G3
$\overline{CD}$

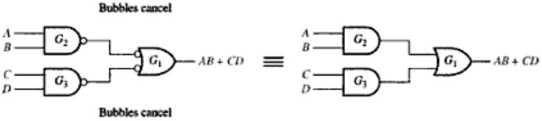
As you can see in Figure 5–19, the output expression, $AB + CD$, is in the form of two AND terms ORed together. This shows that gates G_2 and G_3 act as AND gates and that gate G_1 acts as an OR gate, as illustrated in Figure 5–20(a). This circuit is redrawn in part (b) with NAND symbols for gates G_2 and G_3 and a negative-OR symbol for gate G_1.

Notice in Figure 5–20(b) the bubble-to-bubble connections between the outputs of gates G_2 and G_3 and the inputs of gate G_1. *Since a bubble represents an inversion, two connected bubbles represent a double inversion and therefore cancel each other.* This inversion cancellation

FIGURE 5–20


G2 acts as AND
A
B
G2
G1
AB + CD
C
D
G3
G1 acts as OR
G3 acts as AND

(a) Original NAND logic diagram showing effective gate operation relative to the output expression


Bubbles cancel
A
B
G2
G1
AB + CD
C
D
G3
Bubbles cancel
≡
A
B
G2
G1
AB + CD
C
D
G3

(b) Equivalent NAND/Negative-OR logic diagram

(c) AND-OR equivalent

can be seen in the previous development of the output expression $AB + CD$ and is indicated by the absence of barred terms in the output expression. Thus, the circuit in Figure 5–20(b) is *effectively* an AND-OR circuit as shown in Figure 5–20(c).

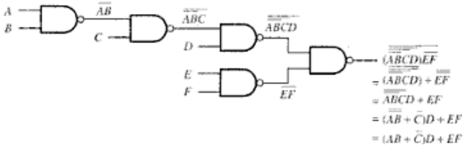
NAND Logic Diagrams

All logic diagrams using NAND gates should be drawn with each gate represented by either a NAND symbol or the equivalent negative-OR symbol to reflect the operation of the gate within the logic circuit. The NAND symbol and the **negative-OR** symbol are called *dual symbols*.

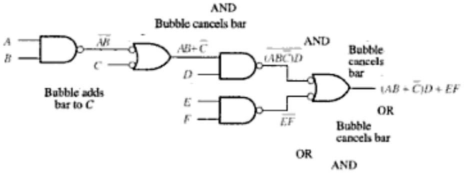
If we begin by representing the output gate with a negative-OR symbol, then we will use the NAND symbol for the level of gates right before the output gate and will alternate the symbols for successive levels of gates as we move away from the output. Always use the gate symbols in such a way that every connection between a gate output and a gate input is either bubble-to-bubble or nonbubble-to-nonbubble. A bubble output should not be connected to a nonbubble input or vice versa in a logic diagram.

Figure 5–21 shows an arrangement of gates to illustrate the procedure of using the appropriate dual symbols for a NAND circuit with several gate levels. Although using all NAND symbols as in Figure 5–21(a) is correct and there is nothing wrong with it, the diagram in part (b) is much easier to "read" and is the preferred method. The shape of the gate indicates the way its inputs will appear in the output expression and thus shows how the gate functions within the logic circuit. For a NAND symbol, the inputs appear ANDed in the output expression; and for a negative-OR symbol, the inputs appear ORed in the output expression, as Figure 5–21(b) illustrates. The dual-symbol diagram in part (b) makes it easier to determine the output expression directly from the logic diagram because each gate symbol indicates the relationship of its input variables as they appear in the output expression.

FIGURE 5–21


A
B
$\overline{AB}$
C
$\overline{\overline{AB}C}$
D
$\overline{\overline{\overline{AB}C}D}$
E
F
$\overline{EF}$
$\overline{(\overline{\overline{\overline{AB}C}D})\overline{EF}}$
$= \overline{(\overline{\overline{\overline{AB}C}D})} + \overline{\overline{EF}}$
$= \overline{\overline{AB}C}D + EF$
$= (\overline{\overline{AB}} + \overline{C})D + EF$
$= (AB + \overline{C})D + EF$

(a) Several Boolean steps are required to arrive at final output expression.

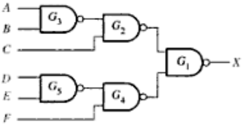

AND
Bubble cancels bar
A
B
$\overline{AB}$
C
$AB+\overline{C}$
AND
$(AB+\overline{C})D$
Bubble cancels bar
D
Bubble adds bar to C
E
F
$\overline{EF}$
$(AB+\overline{C})D+EF$
OR
Bubble cancels bar
OR
AND

(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

EXAMPLE 5–7

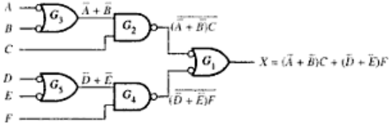
Redraw the logic diagram and develop the output expression for the circuit in Figure 5–22 using the appropriate dual symbols.

▶ FIGURE 5–22


A
B
C
D
E
F
G_3
G_2
G_1
G_5
G_4
X

Solution Redraw the logic diagram in Figure 5–22 with the use of equivalent negative-OR symbols as shown in Figure 5–23. Writing the expression for X directly from the indicated logic operation of each gate gives $X = (\overline{A} + \overline{B})C + (\overline{D} + \overline{E})F$.

▶ FIGURE 5–23


A
B
C
D
E
F
G_3
G_2
G_1
G_5
G_4
$\overline{A} + \overline{B}$
$\overline{(\overline{A} + \overline{B})C}$
$X = (\overline{A} + \overline{B})C + (\overline{D} + \overline{E})F$
$\overline{D} + \overline{E}$
$\overline{(\overline{D} + \overline{E})F}$

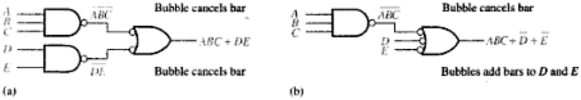
Supplementary Problem Derive the output expression from Figure 5–22 and show it is equivalent to the expression in the solution.

EXAMPLE 5–8

Implement each expression with NAND logic:

(a) $ABC + DE$ **(b)** $ABC + \overline{D} + \overline{E}$

Solution See Figure 5–24.


A
B
C
D
E
$\overline{ABC}$
$\overline{DE}$
Bubble cancels bar
$ABC + DE$
Bubble cancels bar
(a)
A
B
C
D
E
$\overline{ABC}$
Bubble cancels bar
$ABC + \overline{D} + \overline{E}$
Bubbles add bars to D and E
(b)

▲ FIGURE 5–24

Supplementary Problem Convert the NAND circuits in Figure 5–24(a) and (b) to equivalent AND-OR logic.

NOR Logic

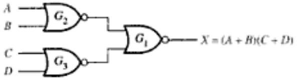
A NOR gate can function as either a NOR or a **negative-AND**, as shown by DeMorgan's theorem.

$$\underset{\text{NOR}}{\overline{A + B}} = \underset{\text{negative-AND}}{\overline{A}\,\overline{B}}$$

Consider the NOR logic in Figure 5–25. The output expression is developed as follows:

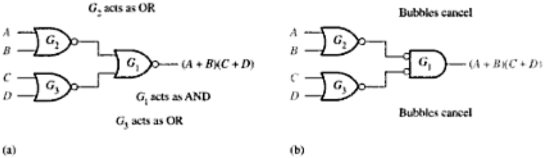
$$X = \overline{\overline{A + B} + \overline{C + D}} = (\overline{\overline{A + B}})(\overline{\overline{C + D}}) = (A + B)(C + D)$$

FIGURE 5–25


A
B
C
D
G_2
G_3
G_1
$X = (A + B)(C + D)$

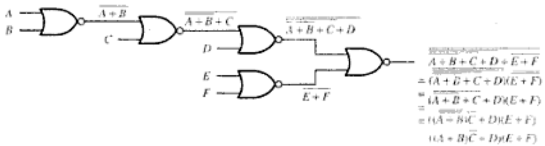
Notice that the output expression $(A + B)(C + D)$ consists of two OR terms ANDed together. This shows that gates G_2 and G_3 act as OR gates and gate G_1 acts as an AND gate, as illustrated in Figure 5–26(a). This circuit is redrawn in part (b) with a negative-AND symbol for gate G_1.

FIGURE 5–26


G_2 acts as OR
A
B
C
D
G_2
G_3
G_1
$(A + B)(C + D)$
G_1 acts as AND
G_3 acts as OR
(a)
Bubbles cancel
A
B
C
D
G_2
G_3
G_1
$(A + B)(C + D)$
Bubbles cancel
(b)

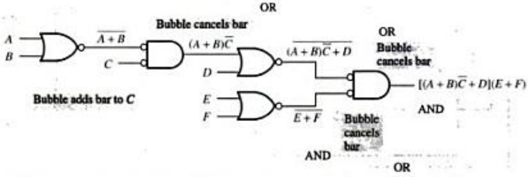
As with NAND logic, the purpose for using the dual symbols is to make the logic diagram easier to read and analyze, as illustrated in the NOR logic circuit in Figure 5–27. When the circuit in part (a) is redrawn with dual symbols in part (b), notice that all output-to-input connections between gates are bubble-to-bubble or nonbubble-to-nonbubble. Again, you can see that the shape of each gate symbol indicates the type of term (AND or OR) that it produces in the output expression, thus making the output expression easier to determine and the logic diagram easier to analyze.

FIGURE 5–27(a)


A
B
C
D
E
F
$\overline{A + B}$
$\overline{\overline{A + B} + C}$
$\overline{\overline{\overline{A + B} + C} + D}$
$\overline{E + F}$
$\overline{\overline{\overline{\overline{A + B} + C} + D} + \overline{E + F}}$
$= (\overline{\overline{A + B} + C} + D)(E + F)$
$= (\overline{\overline{A + B}}\,\overline{C} + D)(E + F)$
$= ((A + B)\overline{C} + D)(E + F)$
$((A + B)\overline{C} + D)(E + F)$

(a) Final output expression is obtained after several Boolean steps.

FIGURE 5–27(b)

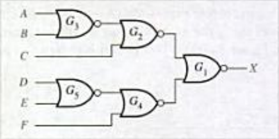

OR
Bubble cancels bar
A
B
$\overline{A+B}$
C
$(A+B)\overline{C}$
D
$\overline{(A+B)\overline{C}+D}$
OR
Bubble cancels bar
$[(A+B)\overline{C}+D](E+F)$
AND
Bubble adds bar to C
E
F
$\overline{E+F}$
Bubble cancels bar
AND
OR

(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

EXAMPLE 5–9

Using appropriate dual symbols, redraw the logic diagram and develop the output expression for the circuit in Figure 5–28.

FIGURE 5–28


A
B
C
D
E
F
G_3
G_2
G_1
G_5
G_4
X

Solution Redraw the logic diagram with the equivalent negative-AND symbols as shown in Figure 5–29. Writing the expression for X directly from the indicated operation of each gate,

$$X = (\overline{A}\,\overline{B} + C)(\overline{D}\,\overline{E} + F)$$

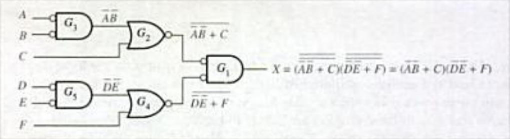

A
B
C
D
E
F
G_3
G_2
G_1
G_5
G_4
$\overline{A}\,\overline{B}$
$\overline{\overline{A}\,\overline{B}+C}$
$\overline{D}\,\overline{E}$
$\overline{\overline{D}\,\overline{E}+F}$
$X = \overline{\overline{(\overline{A}\,\overline{B}+C)}}\;\overline{\overline{(\overline{D}\,\overline{E}+F)}} = (\overline{A}\,\overline{B}+C)(\overline{D}\,\overline{E}+F)$

FIGURE 5–29

Supplementary Problem Prove that the output of the NOR circuit in Figure 5–28 is the same as for the circuit in Figure 5–29.

SECTION 5–4 REVIEW

1. Implement the expression $X = \overline{(\overline{A} + \overline{B} + \overline{C})DE}$ by using NAND logic.
2. Implement the expression $X = \overline{\overline{A}\,\overline{B}\,\overline{C} + (D + E)}$ with NOR logic.

5–5 LOGIC CIRCUIT OPERATION WITH PULSE WAVEFORMS

Several examples of general combinational logic circuits with pulse waveform inputs are examined in this section. Keep in mind that the operation of each gate is the same for pulse inputs as for constant-level inputs. The output of a logic circuit at any given time depends on the inputs at that particular time, so the relationship of the time-varying inputs is of primary importance.

After completing this section, you should be able to

■ Analyze combinational logic circuits with pulse waveform inputs ■ Develop a timing diagram for any given combinational logic circuit with specified inputs

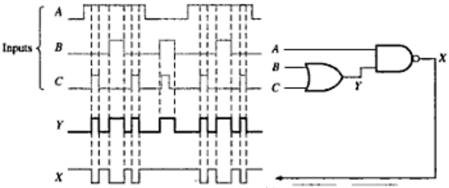
The operation of any gate is the same regardless of whether its inputs are pulsed or constant levels. The nature of the inputs (pulsed or constant levels) does not alter the truth table of a circuit. The examples in this section illustrate the analysis of combinational logic circuits with pulsed inputs.

The following is a review of the operation of individual gates for use in analyzing combinational circuits with pulse waveform inputs:

1. The output of an AND gate is HIGH only when all inputs are HIGH at the same time.
2. The output of an OR gate is HIGH only when at least one of its inputs is HIGH.
3. The output of a NAND gate is LOW only when all inputs are HIGH at the same time.
4. The output of a NOR gate is LOW only when at least one of its inputs is HIGH.

EXAMPLE 5–10

Determine the final output waveform X for the circuit in Figure 5–30, with input waveforms A, B, and C as shown.



$X = \overline{A(B + C)} = \overline{AB + AC}$

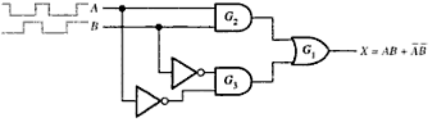
▲ **FIGURE 5–30**

Solution The output expression, $\overline{AB + AC}$, indicates that the output X is LOW when both A and B are HIGH or when both A and C are HIGH or when all inputs are HIGH. The output waveform X is shown in the timing diagram of Figure 5–30. The intermediate waveform Y at the output of the OR gate is also shown.

Supplementary Problem Determine the output waveform if input A is a constant HIGH level.

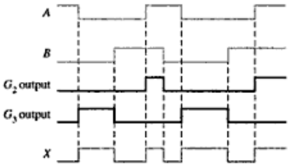
EXAMPLE 5–11

Draw the timing diagram for the circuit in Figure 5–31 showing the outputs of G_1, G_2, and G_3 with the input waveforms, A, and B, as indicated.



▲ **FIGURE 5–31**

Solution When both inputs are HIGH or when both inputs are LOW, the output X is HIGH as shown in Figure 5–32. Notice that this is an exclusive-NOR circuit. The intermediate outputs of gates G_2 and G_3 are also shown in Figure 5–32.

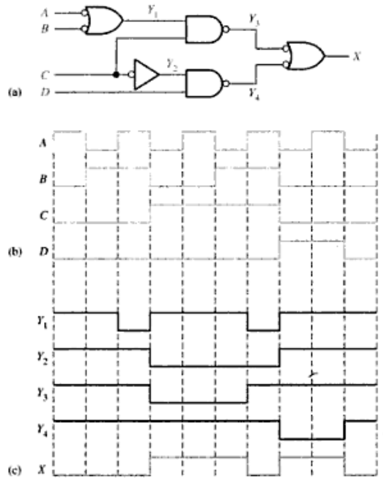


▲ **FIGURE 5–32**

Supplementary Problem Determine the output X in Figure 5–31 if input B is inverted.

EXAMPLE 5–12

Determine the output waveform X for the logic circuit in Figure 5–33(a) by first finding the intermediate waveform at each of points Y_1, Y_2, Y_3, and Y_4. The input waveforms are shown in Figure 5–33(b).


A
B
C
D
Y_1
Y_2
Y_3
Y_4
X
(a)
(b)
(c)

▲ **FIGURE 5–33**

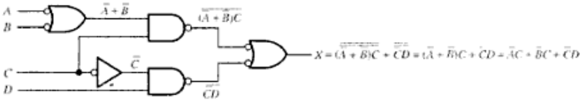
Solution All the intermediate waveforms and the final output waveform are shown in the timing diagram of Figure 5–33(c).

Supplementary Problem Determine the waveforms Y_1, Y_2, Y_3, Y_4 and X if input waveform A is inverted.

EXAMPLE 5–13

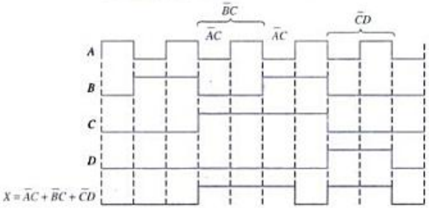
Determine the output waveform X for the circuit in Example 5–12, Figure 5–33(a), directly from the output expression.

Solution The output expression for the circuit is developed in Figure 5–34. The SOP form indicates that the output is HIGH when A is LOW and C is HIGH or when B is LOW and C is HIGH or when C is LOW and D is HIGH.


A
B
C
D
$\overline{A}+\overline{B}$
$\overline{(\overline{A}+\overline{B})C}$
$\overline{C}$
$\overline{\overline{C}D}$
$X = \overline{\overline{(\overline{A}+\overline{B})C}} + \overline{\overline{\overline{C}D}} = (\overline{A}+\overline{B})C + \overline{C}D = \overline{A}C + \overline{B}C + \overline{C}D$

▲ **FIGURE 5–34**

The result is shown in Figure 5–35 and is the same as the one obtained by the intermediate-waveform method in Example 5–12. The corresponding product terms for each waveform condition that results in a HIGH output are indicated.



▲ FIGURE 5–35

Supplementary Problem Repeat this example if all the input waveforms are inverted.

SECTION 5–5 REVIEW

1. One pulse with $t_W = 50\ \mu s$ is applied to one of the inputs of an exclusive-OR circuit. A second positive pulse with $t_W = 10\ \mu s$ is applied to the other input beginning 15 μs after the leading edge of the first pulse. Show the output in relation to the inputs.
2. The pulse waveforms *A* and *B* in Figure 5–30 are applied to the exclusive-NOR circuit in Figure 5–31. Develop a complete timing diagram.

SUMMARY

- AND-OR logic produces an output expression in SOP form.
- AND-OR-Invert logic produces a complemented SOP form, which is actually a POS form.
- The operational symbol for exclusive-OR is ⊕. An exclusive-OR expression can be stated in two equivalent ways:

$$A\overline{B} + \overline{A}B = A \oplus B$$

- To do an analysis of a logic circuit, start with the logic circuit, and develop the Boolean output expression or the truth table or both.
- Implementation of a logic circuit is the process in which you start with the Boolean output expressions or the truth table and develop a logic circuit that produces the output function.
- All NAND or NOR logic diagrams should be drawn using appropriate dual symbols so that bubble outputs are connected to bubble inputs and nonbubble outputs are connected to nonbubble inputs.
- When two negation indicators (bubbles) are connected, they effectively cancel each other.

SELF-TEST

Answers are at the end of the chapter.

1. The output expression for an AND-OR circuit having one AND gate with inputs *A*, *B*, *C*, and *D* and one AND gate with inputs *E* and *F* is

(a) $ABCDEF$ **(b)** $A + B + C + D + E + F$
(c) $(A + B + C + D)(E + F)$ **(d)** $ABCD + EF$

2. A logic circuit with an output $X = A\overline{B}C + A\overline{C}$ consists of

(a) two AND gates and one OR gate
(b) two AND gates, one OR gate, and two inverters
(c) two OR gates, one AND gate, and two inverters
(d) two AND gates, one OR gate, and one inverter

3. To implement the expression $\overline{A}BCD + A\overline{B}CD + AB\overline{C}\overline{D}$, it takes one OR gate and

(a) one AND gate **(b)** three AND gates
(c) three AND gates and four inverters **(d)** three AND gates and three inverters

4. The expression $\overline{A}BCD + ABC\overline{D} + A\overline{B}\overline{C}D$

(a) cannot be simplified
(b) can be simplified to $\overline{A}BC + A\overline{B}$
(c) can be simplified to $ABC\overline{D} + \overline{A}B\overline{C}$
(d) None of these answers is correct.

5. The output expression for an AND-OR-Invert circuit having one AND gate with inputs, *A*, *B*, *C*, and *D* and one AND gate with inputs *E* and *F* is

(a) $ABCD + EF$ **(b)** $\overline{A} + \overline{B} + \overline{C} + \overline{D} + \overline{E} + \overline{F}$
(c) $\overline{(A + B + C + D)(E + F)}$ **(d)** $(\overline{A} + \overline{B} + \overline{C} + \overline{D})(\overline{E} + \overline{F})$

6. An exclusive-OR function is expressed as

(a) $\overline{A}\overline{B} + AB$ **(b)** $\overline{A}B + A\overline{B}$
(c) $(\overline{A} + B)(A + \overline{B})$ **(d)** $(\overline{A} + \overline{B}) + (A + B)$

7. The AND operation can be produced with

(a) two NAND gates **(b)** three NAND gates
(c) one NOR gate **(d)** three NOR gates

8. The OR operation can be produced with

(a) two NOR gates **(b)** three NAND gates
(c) four NAND gates **(d)** both answers (a) and (b)

9. When using dual symbols in a logic diagram,

(a) bubble outputs are connected to bubble inputs
(b) the NAND symbols produce the AND operations
(c) the negative-OR symbols produce the OR operations
(d) All of these answers are true.
(e) None of these answers is true.

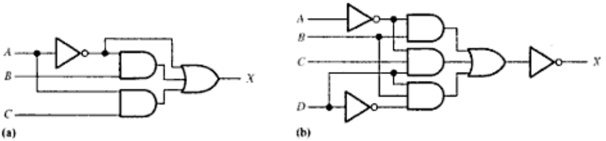
10. All Boolean expressions can be implemented with

(a) NAND gates only
(b) NOR gates only
(c) combinations of NAND and NOR gates
(d) combinations of AND gates, OR gates, and inverters
(e) any of these

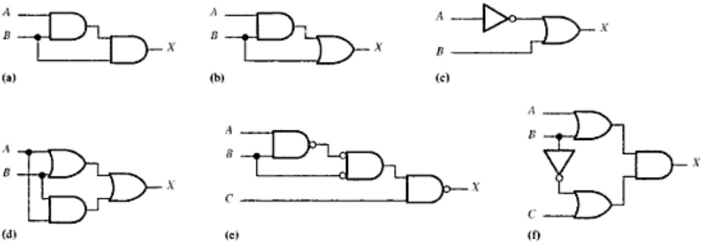
PROBLEMS

SECTION 5–1 Basic Combinational Logic Circuits

1. Draw the ANSI distinctive shape logic diagram for a 3-wide, 4-input AND-OR-Invert circuit. Also, draw the ANSI standard rectangular outline symbol.
2. Write the output expression for each circuit in Figure 5–36.
3. Write the output expression for each circuit as it appears in Figure 5–37.



▲ FIGURE 5–36



▲ FIGURE 5–37

4. Write the output expression for each circuit as it appears in Figure 5–38 and then change each circuit to an equivalent AND-OR configuration.
5. Develop the truth table for each circuit in Figure 5–37.
6. Develop the truth table for each circuit in Figure 5–38.
7. Show that an exclusive-NOR circuit produces a POS output.

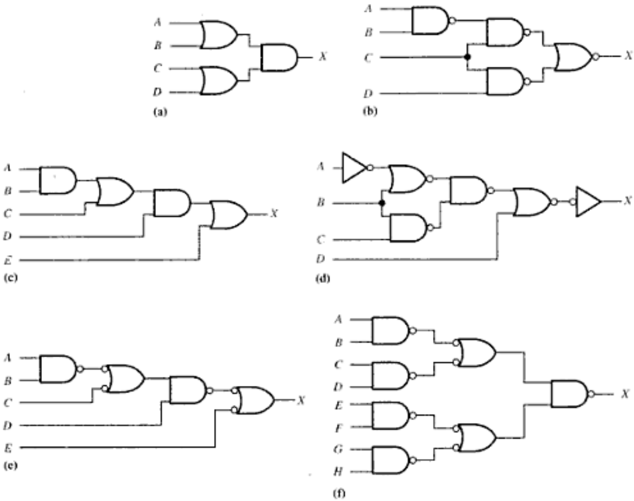
SECTION 5–2 Implementing Combinational Logic

8. Use AND gates, OR gates, or combinations of both to implement the following logic expressions as stated:

(a) $X = AB$ **(b)** $X = A + B$ **(c)** $X = AB + C$

(d) $X = ABC + D$ **(e)** $X = A + B + C$ **(f)** $X = ABCD$

(g) $X = A(CD + B)$ **(h)** $X = AB(C + DEF) + CE(A + B + F)$



▲ **FIGURE 5–38**

9. Use AND gates, OR gates, and inverters as needed to implement the following logic expressions as stated:

(a) $X = AB + \overline{B}C$ **(b)** $X = A(B + \overline{C})$

(c) $X = A\overline{B} + AB$ **(d)** $X = \overline{ABC} + B(EF + \overline{G})$

(e) $X = A[BC(A + B + C + D)]$ **(f)** $X = B(C\overline{D}E + \overline{E}FG)(\overline{AB} + C)$

10. Use NAND gates, NOR gates, or combinations of both to implement the following logic expressions as stated:

(a) $X = \overline{A}B + CD + (\overline{A + B})(ACD + \overline{BE})$ **(b)** $X = AB\overline{C}\overline{D} + D\overline{E}F + \overline{AF}$

(c) $X = \overline{A}[B + \overline{C}(D + E)]$

11. Implement a logic circuit for the truth table in Table 5–6.

12. Implement a logic circuit for the truth table in Table 5–7.

13. Simplify the circuit in Figure 5–39 as much as possible, and verify that the simplified circuit is equivalent to the original by showing that the truth tables are identical.

14. Repeat Problem 13 for the circuit in Figure 5–40.

15. Minimize the gates required to implement the functions in each part of Problem 9 in SOP form.

16. Minimize the gates required to implement the functions in each part of Problem 10 in SOP form.

17. Minimize the gates required to implement the function of the circuit in each part of Figure 5–38 in SOP form.

TABLE 5-6

INPUTS			OUTPUT
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

TABLE 5-7

INPUTS				OUTPUT
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

FIGURE 5-39

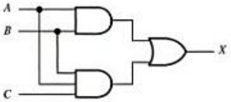
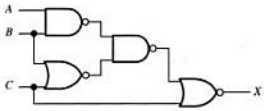


FIGURE 5-40



SECTION 5–3 The Universal Property of NAND and NOR Gates

18. Implement the logic circuits in Figure 5–36 using only NAND gates.

19. Implement the logic circuits in Figure 5–40 using only NAND gates.

20. Repeat Problem 18 using only NOR gates.

21. Repeat Problem 19 using only NOR gates.

SECTION 5–4 Combinational Logic Using NAND and NOR Gates

22. Show how the following expressions can be implemented as stated using only NOR gates:

(a) $X = ABC$ (b) $X = \overline{ABC}$ (c) $X = A + B$

(d) $X = A + B + \overline{C}$ (e) $X = \overline{AB} + \overline{CD}$ (f) $X = (A + B)(C + D)$

(g) $X = AB[C(\overline{DE} + \overline{AB}) + \overline{BCE}]$

23. Repeat Problem 22 using only NAND gates.

24. Implement each function in Problem 8 by using only NAND gates.

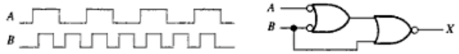
25. Implement each function in Problem 9 by using only NAND gates.

SECTION 5–5 Logic Circuit Operation with Pulse Waveforms

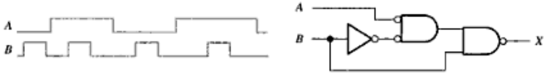
26. Given the logic circuit and the input waveforms in Figure 5–41, draw the output waveform.

27. For the logic circuit in Figure 5–42, draw the output waveform in proper relationship to the inputs.

▶ **FIGURE 5–41**

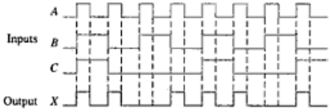


▶ **FIGURE 5–42**



28. For the input waveforms in Figure 5–43, what logic circuit will generate the output waveform shown?

▶ **FIGURE 5–43**



29. Repeat Problem 28 for the waveforms in Figure 5–44.

30. For the circuit in Figure 5–45, draw the waveforms at the numbered points in the proper relationship to each other.

31. Assuming a propagation delay through each gate of 10 nanoseconds (ns), determine if the *desired* output waveform X in Figure 5–46 (a pulse with a minimum $t_W = 25$ ns positioned as shown) will be generated properly with the given inputs.

▶ **FIGURE 5–44**

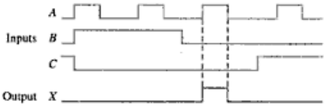


FIGURE 5–45

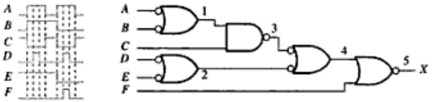
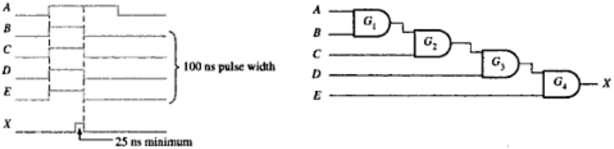


FIGURE 5–46



ANSWERS

SECTION REVIEWS

SECTION 5–1 Basic Combinational Logic Circuits

1. (a) $\overline{AB + CD} = \overline{1 \cdot 0 + 1 \cdot 0} = 1$ (b) $\overline{AB + CD} = \overline{1 \cdot 1 + 0 \cdot 1} = 0$
   (c) $\overline{AB + CD} = \overline{0 \cdot 1 + 1 \cdot 1} = 0$
2. (a) $A\overline{B} + \overline{A}B = 1 \cdot \overline{0} + \overline{1} \cdot 0 = 1$ (b) $A\overline{B} + \overline{A}B = 1 \cdot \overline{1} + \overline{1} \cdot 1 = 0$
   (c) $A\overline{B} + \overline{A}B = 0 \cdot \overline{1} + \overline{0} \cdot 1 = 1$ (d) $A\overline{B} + \overline{A}B = 0 \cdot \overline{0} + \overline{0} \cdot 0 = 0$
3. $X = 1$ when $ABC = 000, 011, 101, 110$, and 111; $X = 0$ when $ABC = 001, 010$, and 100
4. $X = AB + \overline{A}\overline{B}$; the circuit consists of two AND gates, one OR gate, and two inverters. See Figure 5–7(b) for diagram.

SECTION 5–2 Implementing Combinational Logic

1. (a) $X = ABC + AB + AC$: three AND gates, one OR gate
   (b) $X = AB(C + DE)$: three AND gates, one OR gate
2. $X = ABC + \overline{A}\overline{B}\overline{C}$; two AND gates, one OR gate, and three inverters
3. (a) $X = AB(C + 1) + AC = AB + AC$ (b) $X = AB(C + DE) = ABC + ABDE$

SECTION 5–3 The Universal Property of NAND and NOR gates

1. (a) $X = \overline{A} + B$: a 2-input NAND gate with A and $\overline{B}$ on its inputs.
   (b) $X = A\overline{B}$: a 2-input NAND with A and $\overline{B}$ on its inputs, followed by one NAND used as an inverter.
2. (a) $X = \overline{A} + B$: a 2-input NOR with inputs $\overline{A}$ and B, followed by one NOR used as an inverter.
   (b) $X = A\overline{B}$: a 2-input NOR with $\overline{A}$ and B on its inputs

SECTION 5–4 Combinational Logic Using NAND and NOR gates

1. $X = \overline{(\overline{A} + \overline{B} + \overline{C})DE}$: a 3-input NAND with inputs, A, B, and C, with its output connected to a second 3-input NAND with two other inputs, D and E
2. $X = \overline{\overline{A}\overline{B}\overline{C} + (D + E)}$: a 3-input NOR with inputs A, B, and C, with its output connected to a second 3-input NOR with two other inputs, D and E

SECTION 5–5 Logic Circuit Operation with Pulse Waveforms

1. The exclusive-OR output is a 15 μs pulse followed by a 25 μs pulse, with a separation of 10 μs between the pulses.
2. The output of the exclusive-NOR is HIGH when both inputs are HIGH or when both inputs are LOW.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

5–1 $X = AB + AC + BC$

5–2 $X = \overline{AB + AC + BC}$

If $A = 0$ and $B = 0$, $X = \overline{0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1} = \overline{0} = 1$

If $A = 0$ and $C = 0$, $X = \overline{0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0} = \overline{0} = 1$

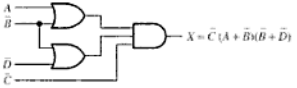
If $B = 0$ and $C = 0$, $X = \overline{1 \cdot 0 + 1 \cdot 0 + 0 \cdot 0} = \overline{0} = 1$

5–3 Cannot be simplified **5–4** Cannot be simplified **5–5** $X = A + B + C + D$ is valid.

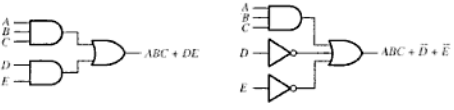
5–6 See Figure 5–47. **5–7** $X = \overline{(\overline{\overline{AB}C})(\overline{\overline{DE}F})} = (\overline{AB})C + (\overline{DE})F = (\overline{A} + \overline{B})C + (\overline{D} + \overline{E})F$

5–8 See Figure 5–48.

▶ FIGURE 5–47


A
$\overline{B}$
$\overline{D}$
$\overline{C}$
$X = \overline{C}(A + \overline{B})(\overline{B} + \overline{D})$

▶ FIGURE 5–48

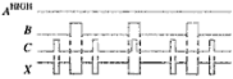

A
B
C
D
E
ABC + DE
A
B
C
D
E
$ABC + \overline{D} + \overline{E}$

(a) (b)

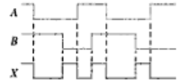
5–9 $X = \overline{\overline{(\overline{A + B} + C)} + \overline{(\overline{D + E} + F)}} = (\overline{A + B} + C)(\overline{D + E} + F) = (\overline{A}\,\overline{B} + C)(\overline{D}\,\overline{E} + F)$

5–10 See Figure 5–49. **5–11** See Figure 5–50.

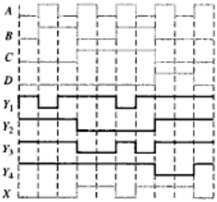
5–12 See Figure 5–51. **5–13** See Figure 5–52.



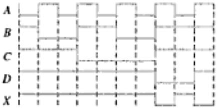
▲ FIGURE 5–49



▲ FIGURE 5–50


A
B
C
D
Y_1
Y_2
Y_3
Y_4
X

▲ FIGURE 5–51



▲ FIGURE 5–52

SELF-TEST

1. (d) **2.** (b) **3.** (c) **4.** (a) **5.** (d) **6.** (b) **7.** (a) **8.** (d)
9. (d) **10.** (e)

6

FUNCTIONS OF COMBINATIONAL LOGIC

CHAPTER OBJECTIVES

- Describe the logic functions of the comparator, adder, code converter, encoder, decoder, multiplexer, demultiplexer, counter, and register
- Distinguish between half-adders and full-adders
- Use full-adders to implement multibit parallel binary adders
- Explain the differences between ripple carry and look-ahead carry parallel adders
- Use the magnitude comparator to determine the relationship between two binary numbers and use cascaded comparators to handle the comparison of larger numbers
- Implement a basic binary decoder
- Use BCD-to-7-segment decoders in display systems
- Apply a decimal-to-BCD priority encoder in a simple keyboard application
- Convert from binary to Gray code, and Gray code to binary by using logic devices
- Apply multiplexers in data selection, multiplexed displays, logic function generation, and simple communications systems
- Use decoders as demultiplexers
- Explain the meaning of parity
- Use parity generators and checkers to detect bit errors in digital systems
- Implement a simple data communications system
- Identify glitches, common bugs in digital systems

INTRODUCTION

In this chapter, several types of fixed-function combinational logic circuits are introduced including adders, comparators, decoders, encoders, code converters, multiplexers (data selectors), demultiplexers, and parity generators/checkers. Examples of system applications of many of these devices are included to demonstrate how the devices can be used in practical situations.

6–1 BASIC OVERVIEW OF LOGIC FUNCTIONS

The three basic logic elements AND, OR, and NOT can be combined to form more complex logic circuits that perform many useful operations and that are used to build complete digital systems. Some of the common logic functions are comparison, arithmetic, code conversion, encoding, decoding, data selection, storage, and counting. This section provides a general overview of these important functions so that you can begin to see how they form the building blocks of digital systems such as computers. The combinational logic circuits do not require memory elements, will be covered in detail in this chapter and the sequential circuits will be covered in detail in later chapters.

After completing this section, you should be able to

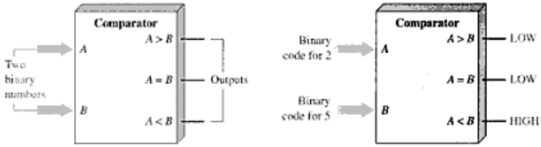
- Identify eight basic types of logic functions
- Describe a basic magnitude comparator
- List the four arithmetic functions
- Describe a basic adder
- Describe a basic encoder
- Describe a basic decoder
- Define multiplexing and demultiplexing
- State how data storage is accomplished
- Describe the function of a basic counter

The Comparison Function

Magnitude comparison is performed by a logic circuit called a **comparator**. A comparator compares two quantities and indicates whether or not they are equal. For example, suppose you have two numbers and wish to know if they are equal or not equal and, if not equal, which is greater. The comparison function is represented in Figure 6–1. One number in binary form (represented by logic levels) is applied to input A, and the other number in binary form (represented by logic levels) is applied to input B. The outputs indicate the relationship of the two numbers by producing a HIGH level on the proper output line. Suppose that a binary representation of the number 2 is applied to input A and a binary representation of the number 5 is applied to input B. (We have discussed the binary representation of numbers and symbols in Chapter 2.) A HIGH level will appear on the $A < B$ (A is less than B) output, indicating the relationship between the two numbers (2 is less than 5). The wide arrows represent a group of parallel lines on which the bits are transferred.

FIGURE 6–1

The comparison function

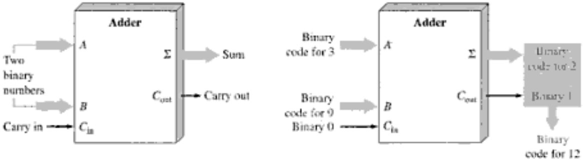

Comparator
A > B
A = B
A < B
A
B
Two binary numbers
Outputs
Comparator
Binary code for 2
Binary code for 5
A
B
A > B
A = B
A < B
LOW
LOW
HIGH

(a) Basic magnitude comparator

(b) Example: A is less than B ($2 < 5$) as indicated by the HIGH output ($A < B$)

The Arithmetic Functions

Addition Addition is performed by a logic circuit called an **adder**. An adder adds two binary numbers (on inputs A and B with a carry input C_{in}) and generates a sum (Σ) and a carry output (C_{out}), as shown in Figure 6–2(a). Figure 6–2(b) illustrates the addition of 3 and 9. You know that the sum is 12; the adder indicates this result by producing 2 on the sum output and 1 on the carry output. Assume that the carry input in this example is 0.


Adder
A
B
C_{in}
Σ
C_{out}
Two binary numbers
Carry in
Sum
Carry out
Adder
Binary code for 3
Binary code for 9
Binary 0
A
B
C_{in}
Σ
C_{out}
Binary code for 2
Binary 1
Binary code for 12

(a) Basic adder

(b) Example: A plus B ($3 + 9 = 12$)

FIGURE 6–2

The addition function

Subtraction Subtraction is also performed by a logic circuit. A **subtracter** requires three inputs: the two numbers that are to be subtracted and a borrow input. The two outputs are the difference and the borrow output. When, for instance, 5 is subtracted from 8 with no borrow input, the difference is 3 with no borrow output. You have seen in Chapter 2 how subtraction can actually be performed by an adder because subtraction is simply a special case of addition.

Multiplication Multiplication is performed by a logic circuit called a *multiplier*. Numbers are always multiplied two at a time, so two inputs are required. The output of the multiplier is the product. Because multiplication is simply a series of additions with shifts in the positions of the partial products, it can be performed by using an adder in conjunction with other circuits.

Division Division can be performed with a series of subtractions, comparisons, and shifts, and thus it can also be done using an adder in conjunction with other circuits. Two inputs to the divider are required, and the outputs generated are the quotient and the remainder.

The Code Conversion Function

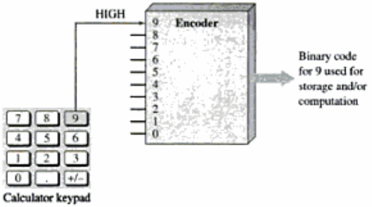
A **code** is a set of bits arranged in a unique pattern and used to represent specified information. A code converter changes one form of coded information into another coded form. Examples are conversion between binary and other codes such as the binary coded decimal (BCD) and the Gray code. Various types of codes have been covered in Chapter 2.

The Encoding Function

The encoding function is performed by a logic circuit called an **encoder**. The encoder converts information, such as a decimal number or an alphabetic character, into some coded form. For example, one certain type of encoder converts each of the decimal digits, 0 through 9, to a binary code. A HIGH level on the input corresponding to a specific decimal digit produces logic levels that represent the proper binary code on the output lines.

Figure 6–3 is a simple illustration of an encoder used to convert (encode) a calculator keystroke into a binary code that can be processed by the calculator circuits.

FIGURE 6–3

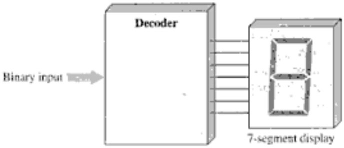


The Decoding Function

The decoding function is performed by a logic circuit called a **decoder**. The decoder converts coded information, such as a binary number, into a noncoded form, such as a decimal form. For example, one particular type of decoder converts a 4-bit binary code into the appropriate decimal digit.

Figure 6–4 is a simple illustration of one type of decoder that is used to activate a 7-segment display. Each of the seven segments of the display is connected to an output line from the decoder. When a particular binary code appears on the decoder inputs, the appropriate output lines are activated and light the proper segments to display the decimal digit corresponding to the binary code.

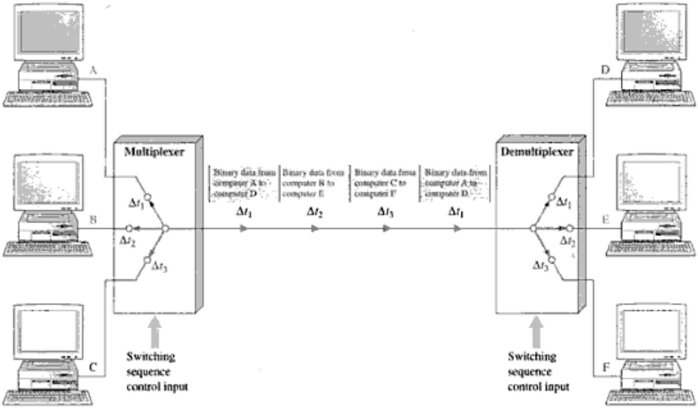
▶ **FIGURE 6–4**


Decoder
Binary input
7-segment display

The Data Selection Function

Two types of circuits that select data are the multiplexer and the demultiplexer. The **multiplexer**, or mux for short, is a logic circuit that switches digital data from several input lines onto a single output line in a specified time sequence. Functionally, a multiplexer can be represented by an electronic switch operation that sequentially connects each of the input lines to the output line. The **demultiplexer** (demux) is a logic circuit that switches digital data from one input line to several output lines in a specified time sequence. Essentially, the demux is a mux in reverse.

Multiplexing and demultiplexing are used when data from several sources are to be transmitted over one line to a distant location and redistributed to several destinations. Figure 6–5 illustrates this type of application where digital data from three computers are sent out along a single line to three other computers at another location.


A
B
C
Multiplexer
Δt_1
Δt_2
Δt_3
Binary data from computer A to computer D
Binary data from computer B to computer E
Binary data from computer C to computer F
Binary data from computer A to computer D
Δt_1
Δt_2
Δt_3
Δt_1
Demultiplexer
Δt_1
Δt_2
Δt_3
D
E
F
Switching sequence control input
Switching sequence control input

▲ **FIGURE 6–5**

In Figure 6–5, binary data from computer A are connected to the output line during time interval Δt_1 and transmitted to the demultiplexer that connects them to computer D. Then, during interval Δt_2, the multiplexer switches to the input from computer B and the demultiplexer switches the output to computer E. During interval Δt_3, the multiplexer switches to the input from computer C and the demultiplexer switches the output to computer F.

To summarize, during the first time interval, computer A sends data to computer D. During the second time interval, computer B sends data to computer E. During the third time interval, computer C sends data to computer F. After this, the first two computers again communicate and the sequence repeats. Because the time is divided up among several sets of systems where each has its turn to send and receive data, this process is called *time division multiplexing* (TDM).

The Storage Function

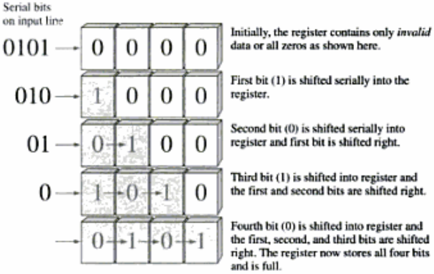
Storage is a function that is required in most digital systems, and its purpose is to retain binary data for a period of time. Some storage devices are used for short-term storage and some are used for long-term storage. A storage device can "memorize" a bit or a group of bits and retain the information as long as necessary. Common types of storage devices are flip-flops, registers, semiconductor memories, magnetic disks, magnetic tape, and optical disks (CDs).

Flip-flops The **flip-flop** is a bistable (two stable states) logic circuit that can store only one bit at a time, either a 1 or a 0. The output of a flip-flop indicates which bit it is storing. A HIGH output indicates that a 1 is stored and a LOW output indicates that a 0 is stored. Flip-flops are implemented with logic gates and are covered in Chapter 7.

Registers A **register** is formed by combining several flip-flops so that groups of bits can be stored. For example, an 8-bit register is constructed from eight flip-flops. In addition to storing bits, registers can be used to shift the bits from one position to another within the register or out of the register to another circuit; therefore, these devices are known as *shift registers.* Shift registers are covered in Chapter 9.

The two basic types of shift registers are serial and parallel. The bits are stored in a serial shift register one at a time, as illustrated in Figure 6–6. A good analogy to the serial shift register is loading passengers onto a bus single file through the door. They also exit the bus single file.

FIGURE 6–6



Initially, the register contains only *invalid* data or all zeros as shown here.

First bit (1) is shifted serially into the register.

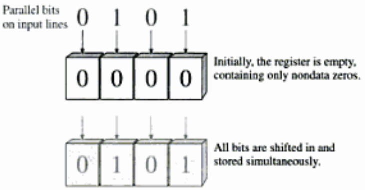
Second bit (0) is shifted serially into register and first bit is shifted right.

Third bit (1) is shifted into register and the first and second bits are shifted right.

Fourth bit (0) is shifted into register and the first, second, and third bits are shifted right. The register now stores all four bits and is full.

The bits are stored in a parallel register simultaneously from parallel lines, as shown in Figure 6–7. For this case, a good analogy is loading passengers on a roller coaster where they enter all of the cars in parallel.

▶ **FIGURE 6–7**

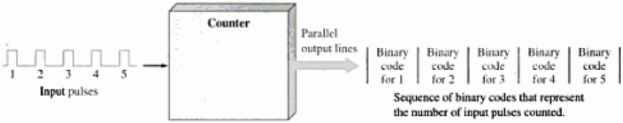

Parallel bits
on input lines
0 1 0 1
0 0 0 0
Initially, the register is empty,
containing only nondata zeros.
0 1 0 1
All bits are shifted in and
stored simultaneously.

Semiconductor Memories Semiconductor memories are devices typically used for storing large numbers of bits. In one type of memory, called the *r*ead-*o*nly *m*emory or ROM, the binary data are permanently or semipermanently stored and cannot be readily changed. In the *r*andom-*a*ccess *m*emory or RAM, the binary data are temporarily stored and can be easily changed. Memories are covered in Chapter 10.

Magnetic Memories Magnetic disk memories are used for mass storage of binary data. Examples are the so-called floppy disks used in computers and the computer's internal hard disk. Magneto-optical disks use laser beams to store and retrieve data. Magnetic tape is still used in memory applications and for backing up data from other storage devices.

The Counting Function

The counting function is very important in digital systems. There are many types of digital **counters**, but their basic purpose is to count events represented by changing levels or pulses. To count, the counter must "remember" the present number so that it can go to the next proper number in sequence. Therefore, storage capability is an important characteristic of all counters, and flip-flops are generally used to implement them. Figure 6–8 illustrates the basic idea of counter operation. Counters are covered in Chapter 8.


Counter
Parallel
output lines
1 2 3 4 5
Input pulses
Binary code for 1
Binary code for 2
Binary code for 3
Binary code for 4
Binary code for 5
Sequence of binary codes that represent
the number of input pulses counted.

▲ **FIGURE 6–8**

Illustration of basic counter operation

SECTION 6–1 REVIEW

1. What does a comparator do?
2. What are the four basic arithmetic operations?
3. Describe encoding and give an example.
4. Describe decoding and give an example.
5. Explain the basic purpose of multiplexing and demultiplexing.
6. Name four types of storage devices.
7. What does a counter do?

6–2 BASIC ADDERS

Adders are important not only in computers but also in many types of digital systems in which numerical data are processed. An understanding of the basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full-adder are introduced.

After completing this section, you should be able to

- Describe the function of a half-adder
- Draw a half-adder logic diagram
- Describe the function of the full-adder
- Draw a full-adder logic diagram using half-adders
- Implement a full-adder using AND-OR logic

The Half-Adder

Recall the basic rules for binary addition as stated in Chapter 2.

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 10 \end{aligned}$$

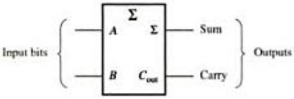
The operations are performed by a logic circuit called a **half-adder**.

> **The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum bit and a carry bit.**

A half-adder is represented by the logic symbol in Figure 6–9.

FIGURE 6–9

Logic symbol for a half-adder



Half-Adder Logic From the operation of the half-adder as stated in Table 6–1, expressions can be derived for the sum and the output carry as functions of the inputs. Notice that the output carry (C_{out}) is a 1 only when both A and B are 1s; therefore, C_{out} can be expressed as the AND of the input variables.

Equation 6–1

$$C_{out} = AB$$

TABLE 6–1

Half-adder truth table

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

Now, observe that the sum output (Σ) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

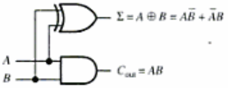
$$\Sigma = A \oplus B$$

Equation 6–2

From Equations 6–1 and 6–2, the logic implementation required for the half-adder function can be developed. The output carry is produced with an AND gate with A and B on the inputs, and the sum output is generated with an exclusive-OR gate, as shown in Figure 6–10. Remember that the exclusive-OR is implemented with AND gates, an OR gate, and inverters.

▶ FIGURE 6–10

Half-adder logic diagram



The Full-Adder

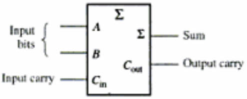
The second category of adder is the **full-adder.**

> **The full-adder accepts two input bits and an input carry and generates a sum output and an output carry.**

The basic difference between a full-adder and a half-adder is that the full-adder accepts an input carry. A logic symbol for a full-adder is shown in Figure 6–11, and the truth table in Table 6–2 shows the operation of a full-adder.

▶ FIGURE 6–11

Logic symbol for a full-adder



▶ TABLE 6–2

Full-adder truth table

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

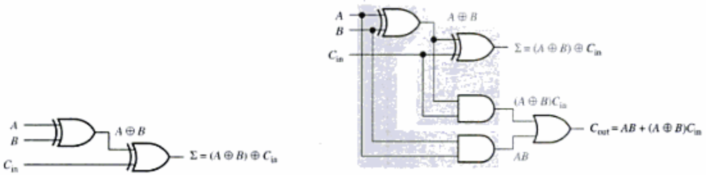
C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

Full-Adder Logic The full-adder must add the two input bits and the input carry. From the half-adder you know that the sum of the input bits A and B is the exclusive-OR of those two variables, $A \oplus B$. For the input carry (C_{in}) to be added to the input bits, it must be exclusive-ORed with $A \oplus B$, yielding the equation for the sum output of the full-adder.

Equation 6–3

$$\Sigma = (A \oplus B) \oplus C_{in}$$

This means that to implement the full-adder sum function, two 2-input exclusive-OR gates can be used. The first must generate the term $A \oplus B$, and the second has as its inputs the output of the first XOR gate and the input carry, as illustrated in Figure 6–12(a).


A
B
A ⊕ B
C_{in}
Σ = (A ⊕ B) ⊕ C_{in}
A
B
A ⊕ B
C_{in}
Σ = (A ⊕ B) ⊕ C_{in}
(A ⊕ B)C_{in}
C_{out} = AB + (A ⊕ B)C_{in}
AB

(a) Logic required to form the sum of three bits

(b) Complete logic circuit for a full-adder (each half-adder is enclosed by a shaded area)

▲ **FIGURE 6–12**

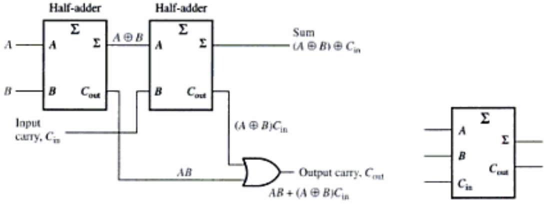
Full-adder logic

The output carry is a 1 when both inputs to the first XOR gate are 1s or when both inputs to the second XOR gate are 1s. You can verify this fact by studying Table 6–2. The output carry of the full-adder is therefore produced by the inputs A ANDed with B and $A \oplus B$ ANDed with C_{in}. These two terms are ORed, as expressed in Equation 6–4. This function is implemented and combined with the sum logic to form a complete full-adder circuit, as shown in Figure 6–4(b).

Equation 6–4

$$C_{out} = AB + (A \oplus B)C_{in}$$

Notice in Figure 6–12(b) there are two half-adders, connected as shown in the block diagram of Figure 6–13(a), with their output carries ORed. The logic symbol shown in Figure 6–13(b) will normally be used to represent the full-adder.


Half-adder
Half-adder
Σ
A
Σ
B
C_{out}
A ⊕ B
Σ
A
Σ
B
C_{out}
A
B
Input carry, C_{in}
Sum (A ⊕ B) ⊕ C_{in}
(A ⊕ B)C_{in}
AB
Output carry, C_{out}
AB + (A ⊕ B)C_{in}
Σ
A
Σ
B
C_{out}
C_{in}

(a) Arrangement of two half-adders to form a full-adder

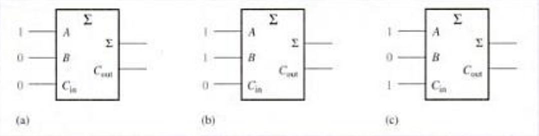
(b) Full-adder logic symbol

▲ **FIGURE 6–13**

Full-adder implemented with half-adders

EXAMPLE 6–13

For each of the three full-adders in Figure 6–14, determine the outputs for the inputs shown.



▲ FIGURE 6–14

Solution **(a)** The input bits are $A = 1$, $B = 0$, and $C_{in} = 0$.

$$1 + 0 + 0 = 1 \text{ with no carry}$$

Therefore, $\mathbf{\Sigma = 1}$ and $\mathbf{C_{out} = 0}$.

(b) The input bits are $A = 1$, $B = 1$, and $C_{in} = 0$.

$$1 + 1 + 0 = 0 \text{ with a carry of } 1$$

Therefore, $\mathbf{\Sigma = 0}$ and $\mathbf{C_{out} = 1}$.

(c) The input bits are $A = 1$, $B = 0$, and $C_{in} = 1$.

$$1 + 0 + 1 = 0 \text{ with a carry of } 1$$

Therefore, $\mathbf{\Sigma = 0}$ and $\mathbf{C_{out} = 1}$.

Supplementary Problem What are the full-adder outputs for $A = 1$, $B = 1$, and $C_{in} = 1$?

SECTION 6–2 REVIEW

Answers are at the end of the chapter.

1. Determine the sum (Σ) and the output carry (C_{out}) of a half-adder for each set of input bits:
   (a) 01 (b) 00 (c) 10 (d) 11
2. A full-adder has $C_{in} = 1$. What are the sum (Σ) and the output carry (C_{out}) when $A = 1$ and $B = 1$?

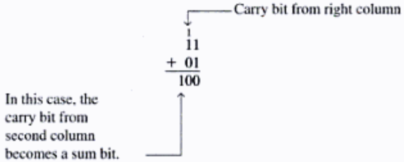
6–3 PARALLEL BINARY ADDERS

Two or more full-adders are connected to form parallel binary adders. In this section, you will learn the basic operation of this type of adder and its associated input and output functions.

After completing this section, you should be able to

- Use full-adders to implement a parallel binary adder
- Explain the addition process in a parallel binary adder
- Use the truth table for a 4-bit parallel adder
- Apply the 74LS83A and the 74LS283 for the addition of two 4-bit numbers
- Expand the 4-bit adder to accommodate 8-bit or 16-bit addition

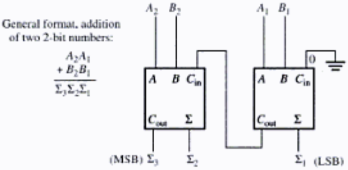
As you saw in Section 6–2, a single full-adder is capable of adding two 1-bit numbers and an input carry. To add binary numbers with more than one bit, additional full-adders must be used. When one binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left, as illustrated here with 2-bit numbers.


Carry bit from right column
1
11
+ 01
100
In this case, the carry bit from second column becomes a sum bit.

To add two binary numbers, a full-adder is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on. The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure 6–15 for a 2-bit adder. Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.

FIGURE 6–15

A 2-bit adder.

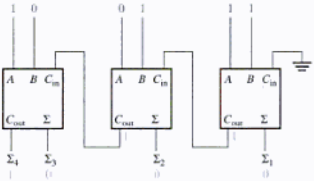

General format, addition of two 2-bit numbers:
A_2A_1
$+ B_2B_1$
$\Sigma_3\Sigma_2\Sigma_1$
A_2
B_2
A_1
B_1
A
B
C_{in}
C_{out}
Σ
0
(MSB) Σ_3
Σ_2
Σ_1 (LSB)

In Figure 6–15 the least significant bits (LSB) of the two numbers are represented by A_1 and B_1. The next higher-order bits are represented by A_2 and B_2. The three sum bits are Σ_1, Σ_2, and Σ_3. Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum, Σ_3.

EXAMPLE 6–2

Determine the sum generated by the 3-bit parallel adder in Figure 6–16 and show the intermediate carries when the binary numbers 101 and 011 are being added.

FIGURE 6–16

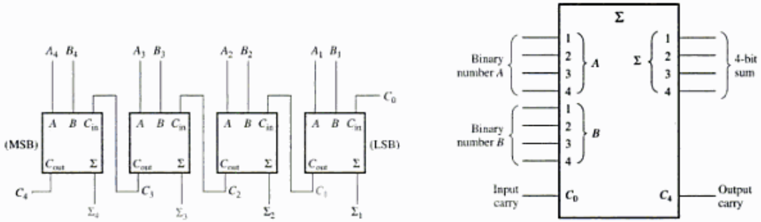

1
0
0
1
1
1
A
B
C_{in}
C_{out}
Σ
1
1
Σ_4
Σ_3
Σ_2
Σ_1
1
0
0
0

Solution The LSBs of the two numbers are added in the right-most full-adder. The sum bits and the intermediate carries are indicated in Figure 6–16.

Supplementary Problem What are the sum outputs when 111 and 101 are added by the 3-bit parallel adder?

Four-Bit Parallel Adders

A group of four bits is called a **nibble**. A basic 4-bit parallel adder is implemented with four full-adder stages as shown in Figure 6–17. Again, the LSBs (A_1 and B_1) in each number being added go into the right-most full-adder; the higher-order bits are applied as shown to the successively higher-order adders, with the MSBs (A_4 and B_4) in each number being applied to the left-most full-adder. The carry output of each adder is connected to the carry input of the next higher-order adder as indicated. These are called *internal carries*.



(a) Block diagram (b) Logic symbol

▲ **FIGURE 6–17**

A 4-bit parallel adder

In keeping with most manufacturers' data sheets, the input labelled C_0 is the input carry to the least significant bit adder; C_4, in the case of four bits, is the output carry of the most significant bit adder; and Σ_1 (LSB) through Σ_4 (MSB) are the sum outputs. The logic symbol is shown in Figure 6–17(b).

In terms of the method used to handle carries in a parallel adder, there are two types: the *ripple carry* adder and the *carry look-ahead* adder. A **ripple carry** adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder). The sum and the output carry of any stage cannot be produced until the input carry occurs; this causes a time delay in the addition process. The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the *A* and *B* inputs are already present.

A method of speeding up the addition process by eliminating this ripple carry delay is called **look-ahead carry** addition. The look-ahead carry adder anticipates the output carry of each stage, and based on the input bits of each stage, produces the output carry by either *carry generation* or *carry propagation*.

Carry generation occurs when an output carry is produced (generated) internally by the full-adder. A carry is generated only when both input bits are 1s. The generated carry, C_g, is expressed as the AND function of the two input bits, *A* and *B*.

$$C_g = AB$$

Carry propagation occurs when the input carry is rippled to become the output carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s. The propagated carry, C_p, is expressed as the OR function of the input bits.

$$C_p = A + B$$

Truth Table for a 4-Bit Parallel Adder

Table 6–3 is the truth table for a 4-bit adder. On some data sheets, truth tables may be called *function tables* or *functional truth tables*. The subscript n represents the adder bits and can be 1, 2, 3, or 4 for the 4-bit adder. C_{n-1} is the carry from the previous adder. Carries C_1, C_2, and C_3 are generated internally. C_0 is an external carry input and C_4 is an output. Example 6–3 illustrates how to use Table 6–3.

▶ **TABLE 6–3**

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EXAMPLE 6–3

Use the 4-bit parallel adder truth table (Table 6–3) to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

$$A_4A_3A_2A_1 = 1100 \quad \text{and} \quad B_4B_3B_2B_1 = 1100$$

Solution For $n = 1$: $A_1 = 0$, $B_1 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_1 = \mathbf{0} \quad \text{and} \quad C_1 = 0$$

For $n = 2$: $A_2 = 0$, $B_2 = 0$, and $C_{n-1} = 0$. From the 1st row of the table,

$$\Sigma_2 = \mathbf{0} \quad \text{and} \quad C_2 = 0$$

For $n = 3$: $A_3 = 1$, $B_3 = 1$, and $C_{n-1} = 0$. From the 4th row of the table,

$$\Sigma_3 = \mathbf{0} \quad \text{and} \quad C_3 = 1$$

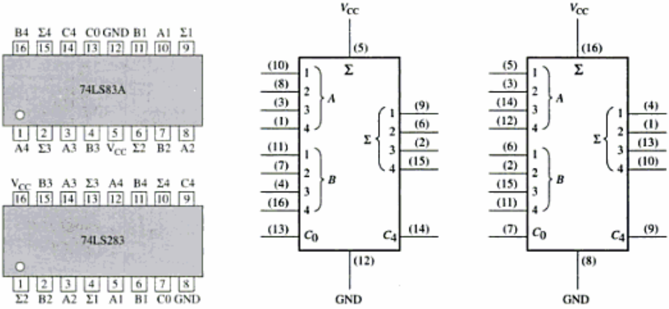
For $n = 4$: $A_4 = 1$, $B_4 = 1$, and $C_{n-1} = 1$. From the last row of the table,

$$\Sigma_4 = \mathbf{1} \quad \text{and} \quad C_4 = \mathbf{1}$$

C_4 becomes the output carry; the sum of 1100 and 1100 is 11000.

Supplementary Problem Use the truth table (Table 6–3) to find the result of adding the binary numbers 1011 and 1010.

Examples of 4-bit parallel adders that are available in IC form are the 74LS83A and the 74LS283 low-power Schottky TTL devices. The 74LS83A and the 74LS283 are functionally identical to each other but not pin compatible; that is, the pin numbers for the inputs and outputs are different due to different power and ground pin connections. For the 74LS83A, V_{CC} is pin 5 and ground is pin 12 on the 16-pin package. For the 74LS283, V_{CC} is pin 16 and ground is pin 8, which is a more standard configuration. Pin diagrams and logic symbols for both of these devices are shown, with pin numbers in parentheses on the logic symbols, in Figure 6–18.


B4 Σ4 C4 C0 GND B1 A1 Σ1
16 15 14 13 12 11 10 9
74LS83A
1 2 3 4 5 6 7 8
A4 Σ3 A3 B3 VCC Σ2 B2 A2
VCC B3 A3 Σ3 A4 B4 Σ4 C4
16 15 14 13 12 11 10 9
74LS283
1 2 3 4 5 6 7 8
Σ2 B2 A2 Σ1 A1 B1 C0 GND
VCC
(5)
(10) (8) (3) (1)
1 2 3 4
A
Σ
(11) (7) (4) (16)
B
(13) C0
Σ 1 2 3 4
(9) (6) (2) (15)
C4 (14)
(12)
GND
VCC
(16)
(5) (3) (14) (12)
A
(6) (2) (15) (11)
B
(7) C0
(4) (1) (13) (10)
C4 (9)
(8)
GND

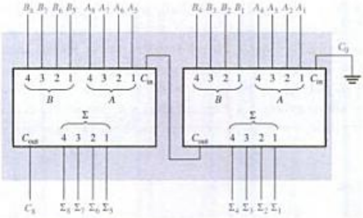
(a) Pin diagrams (b) 74LS83A (c) 74LS283

▲ **FIGURE 6–18**

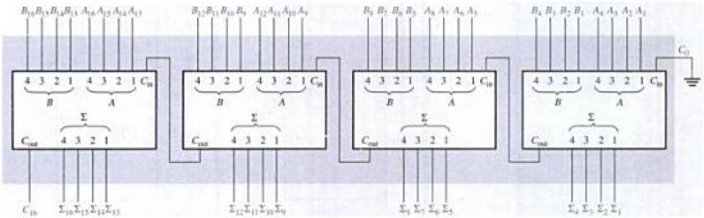
Adder Expansion

The 4-bit parallel adder can be expanded to handle the addition of two 8-bit numbers by using two 4-bit adders. The carry input of the low-order adder (C_0) is connected to ground because there is no carry into the least significant bit position, and the carry output of the low-order adder is connected to the carry input of the high-order adder, as shown in Figure 6–19(a). This process is known as **cascading**. Notice that, in this case, the output carry is designated C_8 because it is generated from the eighth bit position. The low-order adder is the one that adds the lower or less significant four bits in the numbers, and the high-order adder is the one that adds the higher or more significant four bits in the 8-bit numbers.

Similarly, four 4-bit adders can be cascaded to handle two 16-bit numbers as shown in Figure 6–19(b). Notice that the output carry is designated C_{16} because it is generated from the sixteenth bit position.


$B_8\ B_7\ B_6\ B_5\ A_8\ A_7\ A_6\ A_5$
$B_4\ B_3\ B_2\ B_1\ A_4\ A_3\ A_2\ A_1$
C_0
4 3 2 1 4 3 2 1 C_{in}
B
A
Σ
C_{out} 4 3 2 1
C_8
$\Sigma_8\ \Sigma_7\ \Sigma_6\ \Sigma_5$
$\Sigma_4\ \Sigma_3\ \Sigma_2\ \Sigma_1$

(a) Cascading of two 4-bit adders to form an 8-bit adder


$B_{16}B_{15}B_{14}B_{13}\ A_{16}A_{15}A_{14}A_{13}$
$B_{12}B_{11}B_{10}B_9\ A_{12}A_{11}A_{10}A_9$
$B_8\ B_7\ B_6\ B_5\ A_8\ A_7\ A_6\ A_5$
$B_4\ B_3\ B_2\ B_1\ A_4\ A_3\ A_2\ A_1$
C_0
4 3 2 1 4 3 2 1 C_{in}
B
A
Σ
C_{out} 4 3 2 1
C_{16}
$\Sigma_{16}\Sigma_{15}\Sigma_{14}\Sigma_{13}$
$\Sigma_{12}\Sigma_{11}\Sigma_{10}\Sigma_9$
$\Sigma_8\ \Sigma_7\ \Sigma_6\ \Sigma_5$
$\Sigma_4\ \Sigma_3\ \Sigma_2\ \Sigma_1$

(b) Cascading of four 4-bit adders to form a 16-bit adder

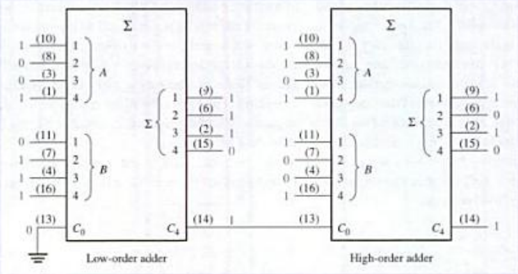
▲ **FIGURE 6–19**

EXAMPLE 6–4

Show how two 74LS83A adders can be connected to form an 8-bit parallel adder. Show output bits for the following 8-bit input numbers:

Solution $A_8A_7A_6A_5A_4A_3A_2A_1 = 10111001$ and $B_8B_7B_6B_5B_4B_3B_2B_1 = 10011110$

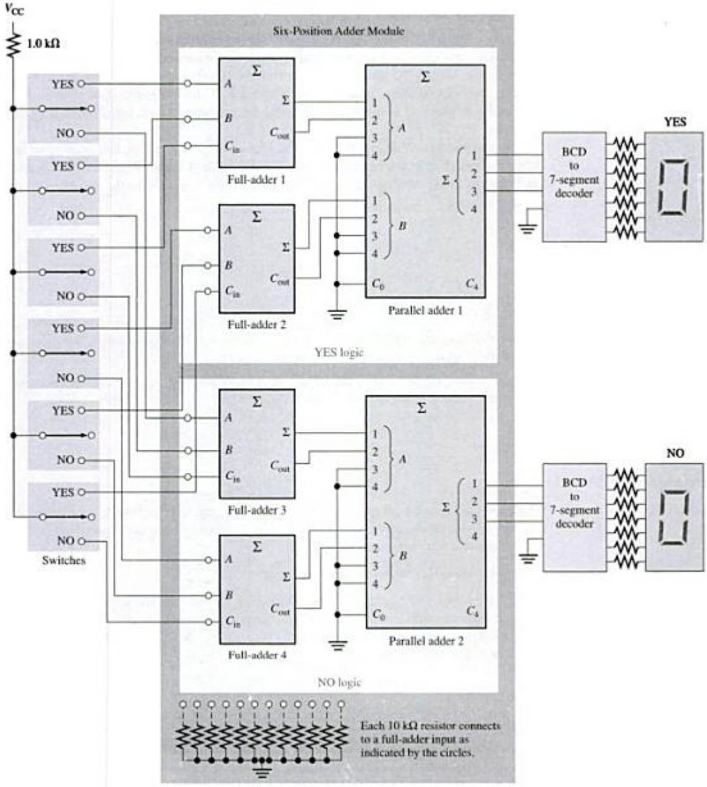
► **FIGURE 6–20**


Σ
1 (10) 1
0 (8) 2
0 (3) 3
1 (1) 4
A
0 (11) 1
1 (7) 2
1 (4) 3
1 (16) 4
B
Σ 1 (9) 1
2 (6) 1
3 (2) 1
4 (15) 0
0 (13) C_0
C_4 (14) 1
Low-order adder
Σ
1 (10) 1
1 (8) 2
0 (3) 3
1 (1) 4
A
1 (11) 1
0 (7) 2
0 (4) 3
1 (16) 4
B
Σ 1 (9) 1
2 (6) 0
3 (2) 1
4 (15) 0
(13) C_0
C_4 (14) 1
High-order adder

Two 74LS83A 4-bit parallel adders are used to implement the 8-bit adder. The only connection between the two 74LS83As is the carry output (pin 14) of the low-order adder to the carry input (pin 13) of the high-order adder, as shown in Figure 6–20. Pin 13 of the low-order adder is grounded (no carry input).

Supplementary Problem Use 74LS283 adders to implement a 12-bit parallel adder.

Application Example



V_{CC}
1.0 kΩ
YES
NO
YES
NO
YES
NO
YES
NO
YES
NO
YES
NO
Switches
Six-Position Adder Module
Σ
A
B
C_{in}
Σ
C_{out}
Full-adder 1
Full-adder 2
Full-adder 3
Full-adder 4
Parallel adder 1
Parallel adder 2
1
2
3
4
A
B
C_0
C_4
YES logic
NO logic
BCD to 7-segment decoder
YES
NO
Each 10 kΩ resistor connects to a full-adder input as indicated by the circles.

▲ FIGURE 6–21

An example of full-adder and parallel adder application is a simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes. For example, this type of system can be used where a group of people are assembled and there is a need for immediately determining opinions (for or against), making decisions, or voting on certain issues or other matters.

In its simplest form, the system includes a switch for "yes" or "no" selection at each position in the assembly and a digital display for the number of yes votes and one for the number of no votes. The basic system is shown in Figure 6–21 for a 6-position setup, but it can be expanded to any number of positions with additional 6-position modules and additional parallel adder and display circuits.

In Figure 6–21 each full-adder can produce the sum of up to three votes. The sum and output carry of each full-adder then goes to the two lower-order inputs of a parallel binary adder. The two higher-order inputs of the parallel adder are connected to ground (0) because there is never a case where the binary input exceeds 0011 (decimal 3). For this basic 6-position system, the outputs of the parallel adder go to a BCD-to-7-segment decoder that drives the 7-segment display. As mentioned, additional circuits must be included when the system is expanded.

The resistors from the inputs of each full-adder to ground assure that each input is LOW when the switch is in the neutral position (CMOS logic is used). When a switch is moved to the "yes" or to the "no" position, a HIGH level (V_{CC}) is applied to the associated full-adder input.

SECTION 6–3 REVIEW

1. Two 4-bit numbers (1101 and 1011) are applied to a 4-bit parallel adder. The input carry is 1. Determine the sum (Σ) and the output carry.
2. How many 74LS283 adders would be required to add two binary numbers each representing decimal numbers up through 1000_{10}?

6–4 COMPARATORS

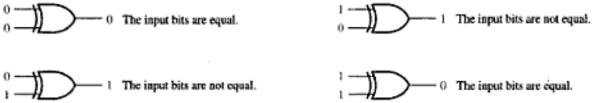
The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities. In its simplest form, a comparator circuit determines whether two numbers are equal.

After completing this section, you should be able to

- Use the exclusive-OR gate as a basic comparator
- Analyze the internal logic of a magnitude comparator that has both equality and inequality outputs
- Apply the 74HC85 comparator to compare the magnitudes of two 4-bit numbers
- Cascade 74HC85s to expand a comparator to eight or more bits

Equality

As you learned in Chapter 3, the exclusive-OR gate can be used as a basic comparator because its output is a 1 if the two input bits are not equal and a 0 if the input bits are equal. Figure 6–22 shows the exclusive-OR gate as a 2-bit comparator.

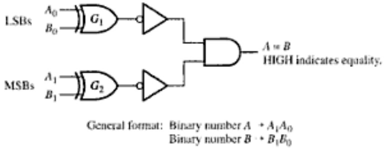

0
0
0 The input bits are equal.
1
0
1 The input bits are not equal.
0
1
1 The input bits are not equal.
1
1
0 The input bits are equal.

▲ **FIGURE 6–22**

Basic comparator operation

In order to compare binary numbers containing two bits each, an additional exclusive-OR gate is necessary. The two least significant bits (LSBs) of the two numbers are compared by gate G_1, and the two most significant bits (MSBs) are compared by gate G_2, as shown in Figure 6–23. If the two numbers are equal, their corresponding bits are the same, and the output of each exclusive-OR gate is a 0. If the corresponding sets of bits are not equal, a 1 occurs on that exclusive-OR gate output.

► **FIGURE 6–23**


LSBs
A_0
B_0
G_1
MSBs
A_1
B_1
G_2
$A = B$
HIGH indicates equality.
General format: Binary number $A \rightarrow A_1A_0$
Binary number $B \rightarrow B_1B_0$

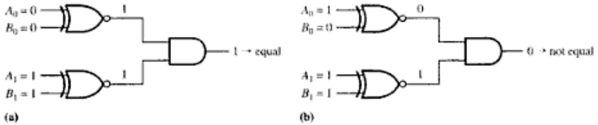
In order to produce a single output indicating an equality or inequality of two numbers, two inverters and an AND gate can be used, as shown in Figure 6–23. The output of each exclusive-OR gate is inverted and applied to the AND gate input. When the two input bits for each exclusive-OR are equal, the corresponding bits of the numbers are equal, producing a 1 on both inputs to the AND gate and thus a 1 on the output. When the two numbers are not equal, one or both sets of corresponding bits are unequal, and a 0 appears on at least one input to the AND gate to produce a 0 on its output. Thus, the output of the AND gate indicates equality (1) or inequality (0) of the two numbers.

Example 6–5 illustrates this operation for two specific cases. The exclusive-OR gate and inverter are replaced by an exclusive-NOR symbol.

EXAMPLE 6–5

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6–24, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 **(b)** 11 and 10


$A_0 = 0$
$B_0 = 0$
1
1 → equal
$A_1 = 1$
$B_1 = 1$
1
(a)
$A_0 = 1$
$B_0 = 0$
0
0 → not equal
$A_1 = 1$
$B_1 = 1$
1
(b)

▲ **FIGURE 6–24**

Solution **(a)** The output is **1** for inputs 10 and 10, as shown in Figure 6–24(a).

(b) The output is **0** for inputs 11 and 10, as shown in Figure 6–24(b).

Supplementary Problem Repeat the process for binary inputs of 01 and 10.

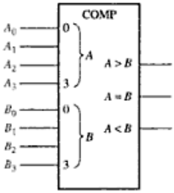
As you know from Chapter 3, the basic comparator can be expanded to any number of bits. The AND gate sets the condition that all corresponding bits of the two numbers must be equal if the two numbers themselves are equal.

Inequality

In addition to the equality output, many IC comparators provide additional outputs that indicate which of the two binary numbers being compared is the larger. That is, there is an output that indicates when number A is greater than number B ($A > B$) and an output that indicates when number A is less than number B ($A < B$), as shown in the logic symbol for a 4-bit comparator in Figure 6–25.

FIGURE 6–25

Logic symbol for a 4-bit comparator.



To determine an inequality of binary numbers A and B, you first examine the highest-order bit in each number. The following conditions are possible:

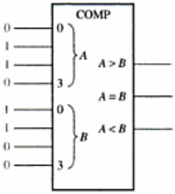
1. If $A_3 = 1$ and $B_3 = 0$, number A is greater than number B.
2. If $A_3 = 0$ and $B_3 = 1$, number A is less than number B.
3. If $A_3 = B_3$, then you must examine the next lower bit position for an inequality.

These three operations are valid for each bit position in the numbers. The general procedure used in a comparator is to check for an inequality in a bit position, starting with the highest-order bits (MSBs). When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored because it is possible for an opposite indication to occur; *the highest-order indication must take precedence.*

EXAMPLE 6–6

Determine the $A = B$, $A > B$, and $A < B$ outputs for the input numbers shown on the comparator in Figure 6–26.

▶ FIGURE 6–26


COMP
0
1
1
0
0
A
3
A > B
A = B
1
1
0
0
0
B
3
A < B

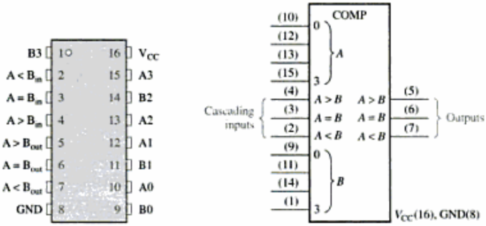
Solution The number on the A inputs is 0110 and the number on the B inputs is 0011. The **$A > B$ output is HIGH and the other outputs ares LOW.**

Supplementary Problem What are the comparator outputs when $A_3A_2A_1A_0 = 1001$ and $B_3B_2B_1B_0 = 1010$?

A 4-BIT MAGNITUDE COMPARATOR

The 74HC85 is a comparator that is also available in other IC families. The pin diagram and logic symbol are shown in Figure 6–27. Notice that this device has all the inputs and outputs of the generalized comparator previously discussed and, in addition, has three cascading inputs: $A < B, A = B, A > B$. These inputs allow several comparators to be cascaded for comparison of any number of bits greater than four. To expand the comparator, the $A < B$, $A = B$, and $A > B$ outputs of the lower-order comparator are connected to the corresponding cascading inputs of the next higher-order comparator. The lowest-order comparator must have a HIGH on the $A = B$ input and LOWs on the $A < B$ and $A > B$ inputs.

▶ FIGURE 6–27

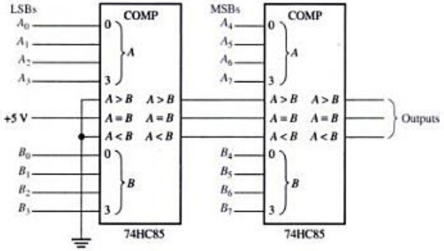

B3
A < B_in
A = B_in
A > B_in
A > B_out
A = B_out
A < B_out
GND
1
2
3
4
5
6
7
8
16
15
14
13
12
11
10
9
V_CC
A3
B2
A2
A1
B1
A0
B0
COMP
(10)
(12)
(13)
(15)
0
A
3
Cascading inputs
(4)
(3)
(2)
A > B
A = B
A < B
A > B
A = B
A < B
(5)
(6)
(7)
Outputs
(9)
(11)
(14)
(1)
0
B
3
V_CC(16), GND(8)

(a) Pin diagram (b) Logic symbol

EXAMPLE 6–7

Use 74HC85 comparators to compare the magnitudes of two 8-bit numbers. Show the comparators with proper interconnections.

Solution Two 74HC85s are required to compare two 8-bit numbers. They are connected as shown in Figure 6–28, in a cascaded arrangement.



▶ **FIGURE 6–28**

An 8-bit magnitude comparator using two 74HC85s

Supplementary Problem Expand the circuit in Figure 6–28 to a 16-bit comparator.

SECTION 6–4 REVIEW

1. The binary numbers $A = 1011$ and $B = 1010$ are applied to the inputs of a 74HC85. Determine the outputs.
2. The binary numbers $A = 11001011$ and $B = 11010100$ are applied to the 8-bit comparator in Figure 6–28. Determine the states of output pins 5, 6, and 7 on each 74HC85.

6–5 DECODERS

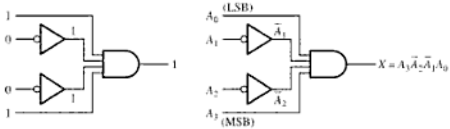
The basic function of a decoder is to detect the presence of a specified combination of bits (code) on its inputs and to indicate the presence of that code by a specified output level. In its general form, a decoder has n input lines to handle n bits and from one to 2^n output lines to indicate the presence of one or more n-bit combinations. In this section, several decoders are introduced. The basic principles can be extended to other types of decoders.

After completing this section, you should be able to

■ Define *decoder* ■ Design a logic circuit to decode any combination of bits
■ Describe the 74HC154 binary-to-decimal decoder ■ Describe the 74HC42 BCD-to-decimal decoder ■ Expand decoders to accommodate larger numbers of bits in a code
■ Describe the 74LS47 BCD-to-7-segment decoder ■ Discuss zero suppression in 7-segment displays ■ Apply decoders to specific applications

The Basic Binary Decoder

Suppose, you need to determine when a binary 1001 occurs on the inputs of a digital circuit. An AND gate can be used as the basic decoding element because it produces a HIGH output only when all of its inputs are HIGH. Therefore, you must make sure that all of the inputs to the AND gate are HIGH when the binary number 1001 occurs; this can be done by inverting the two middle bits (the 0s), as shown in Figure 6–29.



(a) (b)

▲ **FIGURE 6–29**

The logic equation for the **decoder** of Figure 6–29(a) is developed as illustrated in Figure 6–29(b). You should verify that the output is 0 except when $A_0 = 1$, $A_1 = 0$, $A_2 = 0$, and $A_3 = 1$ are applied to the inputs. A_0 is the LSB and A_3 is the MSB. *In the representation of a binary number or other weighted code in this book, the LSB is the right-most bit in a horizontal arrangement and the top-most bit in a vertical arrangement, unless specified otherwise.*

If a NAND gate is used in place of the AND gate in Figure 6–29, a LOW output will indicate the presence of the proper binary code, which is 1001 in this case.

EXAMPLE 6–8

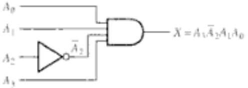
Determine the logic required to decode the binary number 1011 by producing a HIGH level on the output.

Solution The decoding function can be formed by complementing only the variables that appear as 0 in the desired binary number, as follows:

$$X = A_3\bar{A}_2A_1A_0 \qquad (1011)$$

This function can be implemented by connecting the true (uncomplemented) variables A_0, A_1, and A_3 directly to the inputs of an AND gate, and inverting the variable A_2 before applying it to the AND gate input. The decoding logic is shown in Figure 6–30.

▶ **FIGURE 6–30**



Supplementary Problem Develop the logic required to detect the binary code 10010 and produce an active-LOW output.

The 4-Bit Decoder

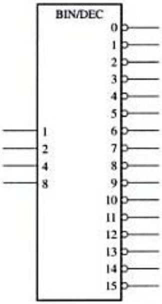
In order to decode all possible combinations of four bits, sixteen decoding gates are required ($2^4 = 16$). This type of decoder is commonly called either a *4-line-to-16-line decoder* because there are four inputs and sixteen outputs or a *1-of-16 decoder* because for any given code on the inputs, one of the sixteen outputs is activated. A list of the sixteen binary codes and their corresponding decoding functions is given in Table 6–4.

If an active-LOW output is required for each decoded number, the entire decoder can be implemented with NAND gates and inverters. In order to decode each of the sixteen binary codes, sixteen NAND gates are required (AND gates can be used to produce active-HIGH outputs).

▼ TABLE 6–4

DECIMAL	BINARY INPUTS				DECODING	OUTPUTS															
DIGIT	A_3	A_2	A_1	A_0	FUNCTION	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

▸ FIGURE 6–31

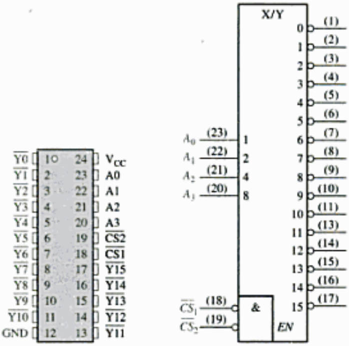


A logic symbol for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs is shown in Figure 6–31. The BIN/DEC label indicates that a binary input makes the corresponding decimal output active. The input labels 8, 4, 2, and 1 represent the binary weights of the input bits ($2^3 2^2 2^1 2^0$).

A 1-OF-16 DECODER

The 74HC154 is a good example of an IC decoder. The logic symbol is shown in Figure 6–32. There is an enable function (*EN*) provided on this device, which is implemented with a NOR gate used as a negative-AND. A LOW level on each chip select input, $\overline{CS}_1$ and $\overline{CS}_2$, is required in order to make the enable gate output (*EN*) HIGH. The enable gate output is connected to an input of *each* NAND gate in the decoder, so it must be HIGH for the NAND gates to be enabled. If the enable gate is not activated by a LOW on both inputs, then all sixteen decoder outputs (*Y*) will be HIGH regardless of the states of the four input variables, A_0, A_1, A_2, and A_3.

FIGURE 6–32



$\overline{Y0}$ 1 24 V_{CC}
$\overline{Y1}$ 2 23 A0
$\overline{Y2}$ 3 22 A1
$\overline{Y3}$ 4 21 A2
$\overline{Y4}$ 5 20 A3
$\overline{Y5}$ 6 19 $\overline{CS2}$
$\overline{Y6}$ 7 18 $\overline{CS1}$
$\overline{Y7}$ 8 17 $\overline{Y15}$
$\overline{Y8}$ 9 16 $\overline{Y14}$
$\overline{Y9}$ 10 15 $\overline{Y13}$
$\overline{Y10}$ 11 14 $\overline{Y12}$
GND 12 13 $\overline{Y11}$
X/Y
A_0 (23) 1
A_1 (22) 2
A_2 (21) 4
A_3 (20) 8
0 (1)
1 (2)
2 (3)
3 (4)
4 (5)
5 (6)
6 (7)
7 (8)
8 (9)
9 (10)
10 (11)
11 (13)
12 (14)
13 (15)
14 (16)
15 (17)
$\overline{CS}_1$ (18)
$\overline{CS}_2$ (19)
&
EN

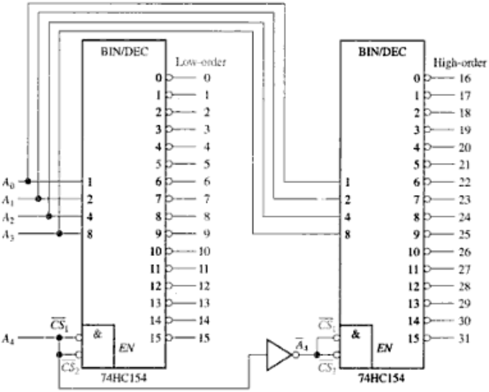
(a) Pin diagram (b) Logic symbol

EXAMPLE 6–9

A certain application requires that a 5-bit number be decoded. Use 74HC154 decoders to implement the logic. The binary number is represented by the format $A_4A_3A_2A_1A_0$.

Solution Since the 74HC154 can handle only four bits, two decoders must be used to decode five bits. The fifth bit, A_4, is connected to the chip select inputs, $\overline{CS}_1$ and $\overline{CS}_2$, of one decoder, and $\overline{A}_4$ is connected to the $\overline{CS}_1$ and $\overline{CS}_2$ inputs of the other decoder, as shown in Figure 6–33. When the decimal number is 15 or less, $A_4 = 0$, and the low-order decoder is enabled and the high-order decoder is disabled. When the decimal number is greater than 15, $A_4 = 1$ so $\overline{A}_4 = 0$, and the high-order decoder is enabled and the low-order decoder is disabled.

FIGURE 6–33



Supplementary Problem Determine the output in Figure 6–33 that is activated for the binary input 10110.

Application Example

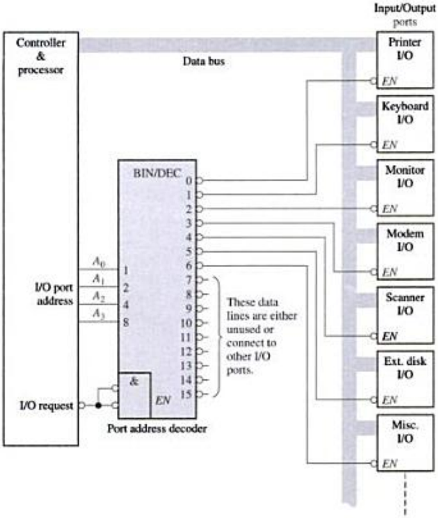
Decoders are used in many types of applications. One example is in computers for input/output selection as depicted in the general diagram of Figure 6–34.

Computers must communicate with a variety of external devices called *peripherals* by sending and/or receiving data through what is known as input/output (I/O) ports. These external devices include printers, modems, scanners, external disk drives, keyboard, video monitors, and other computers. As indicated in Figure 6–34, a decoder is used to select the I/O port as determined by the computer so that data can be sent or received from a specific external device.

Each I/O port has a number, called an address, which uniquely identifies it. When the computer wants to communicate with a particular device, it issues the appropriate address code for the I/O port to which that particular device is connected. This binary port address is decoded and the appropriate decoder output is activated to enable the I/O port.

As shown in Figure 6–34, binary data are transferred within the computer on a data bus, which is a set of parallel lines. For example, an 8-bit bus consists of eight parallel lines that can carry one byte of data at a time. The data bus goes to all of the I/O ports, but any data coming in or going out will only pass through the port that is enabled by the port address decoder.

FIGURE 6–34


Input/Output ports
Controller & processor
Data bus
Printer I/O
EN
Keyboard I/O
EN
Monitor I/O
EN
Modem I/O
EN
Scanner I/O
EN
Ext. disk I/O
EN
Misc. I/O
EN
BIN/DEC
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A_0
A_1
A_2
A_3
1
2
4
8
I/O port address
&
EN
I/O request
Port address decoder
These data lines are either unused or connect to other I/O ports.

The BCD-to-Decimal Decoder

The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications. It is frequently referred as a *4-line-to-10-line decoder* or a *1-of-10 decoder*.

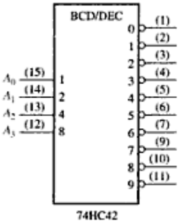
TABLE 6–5

DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0	DECODING FUNCTION
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$

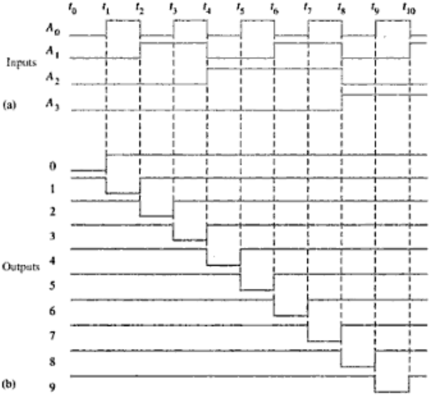
The method of implementation is the same as for the 1-of-16 decoder previously discussed, except that only ten decoding gates are required because the BCD code represents only the ten decimal digits 0 through 9. A list of the ten BCD codes and their corresponding decoding functions is given in Table 6–5. Each of these decoding functions is implemented with NAND gates to provide active-LOW outputs. If an active-HIGH output is required, AND gates are used for decoding. The logic is identical to that of the first ten decoding gates in the 1-of-16 decoder (see Table 6–4).

EXAMPLE 6–10

The 74HC42 is an integrated circuit BCD-to-decimal decoder. The logic symbol is shown in Figure 6–35. If the input waveforms in Figure 6–36(a) are applied to the inputs of the 74HC42, show the output waveforms.



▲ **FIGURE 6–35**



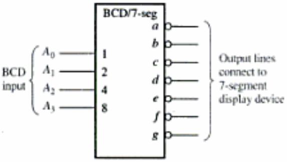
▲ **FIGURE 6–36**

Solution **The output waveforms are shown in Figure 6–36(b). As you can see, the inputs are sequenced through the BCD for digits 0 through 9. The output waveforms in the timing diagram indicate that sequence.**

Supplementary Problem **Construct a timing diagram showing input and output waveforms for the case where the binary inputs sequence through the decimal numbers as follows: 0, 2, 4, 6, 8, 1, 3, 5, and 9.**

The BCD-to-7-Segment Decoder

As you learned in the system application of Chapter 4, the BCD-to-7-segment decoder accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The logic diagram for a basic 7-segment decoder is shown in Figure 6–37.

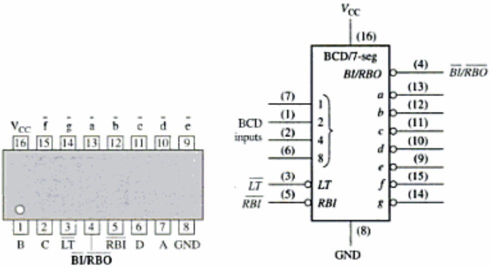

BCD/7-seg
BCD input
A_0
A_1
A_2
A_3
1
2
4
8
a
b
c
d
e
f
g
Output lines connect to 7-segment display device

▲ **FIGURE 6–37**

A BCD-TO-7-SEGMENT DECODER/DRIVER

The 74LS47 is an example of an IC device that decodes a BCD input and drives a 7-segment display. In addition to its decoding and segment drive capability, the 74LS47 has several additional features as indicated by the $\overline{LT}$, $\overline{RBI}$, $\overline{BI}/\overline{RBO}$ functions in the logic symbol of Figure 6–38. As indicated by the bubbles on the logic symbol, all of the outputs (*a* through *g*) are active-LOW as are the $\overline{LT}$ (lamp test), $\overline{RBI}$ (ripple blanking input), and $\overline{BI}/\overline{RBO}$ (blanking input/ripple blanking output) functions. The outputs can drive a common-anode 7-segment display directly. Recall that 7-segment displays were discussed in Chapter 4.

▶ **FIGURE 6–38**


V_{CC}
$\overline{f}$
$\overline{g}$
$\overline{a}$
$\overline{b}$
$\overline{c}$
$\overline{d}$
$\overline{e}$
16
15
14
13
12
11
10
9
1
2
3
4
5
6
7
8
B
C
$\overline{LT}$
$\overline{RBI}$
D
A
GND
$\overline{BI}/\overline{RBO}$
V_{CC}
(16)
BCD/7-seg
BI/RBO
(4)
$\overline{BI}/\overline{RBO}$
BCD inputs
(7)
(1)
(2)
(6)
1
2
4
8
a
b
c
d
e
f
g
(13)
(12)
(11)
(10)
(9)
(15)
(14)
$\overline{LT}$
$\overline{RBI}$
(3)
(5)
LT
RBI
(8)
GND

(a) Pin diagram

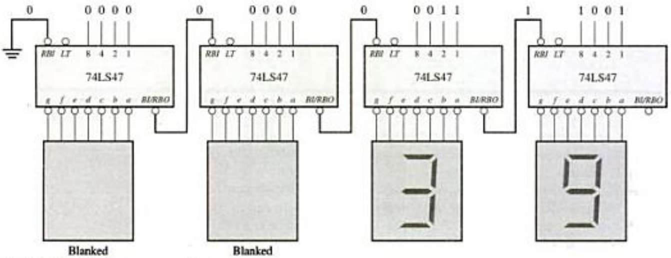
(b) Logic symbol

In addition to decoding a BCD input and producing the appropriate 7-segment outputs, the 74LS47 has lamp test and zero suppression capability.

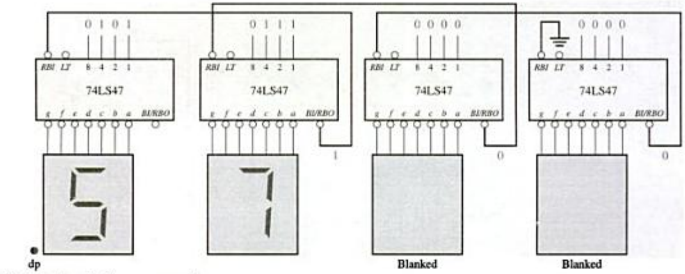
Lamp Test When a LOW is applied to the $\overline{LT}$ input and the $\overline{BI}/\overline{RBO}$ is HIGH, all of the 7 segments in the display are turned on. Lamp test is used to verify that no segments are burned out.

Zero Suppression **Zero suppression** is a feature used for multidigit displays to blank out unnecessary zeros. For example, in a 6-digit display the number 6.4 may be displayed as 006.400 if the zeros are not blanked out. Blanking the zeros at the front of a number is called *leading zero suppression* and blanking the zeros at the back of the number is called *trailing zero suppression*. Keep in mind that only nonessential zeros are blanked. With zero suppression, the number 030.080 will be displayed as 30.08 (the essential zeros remain).

Zero suppression in the 74LS47 is accomplished using the $\overline{RBI}$ and $\overline{BI}/\overline{RBO}$ functions. $\overline{RBI}$ is the ripple blanking input and $\overline{RBO}$ is the ripple blanking output on the 74LS47; these are used for zero suppression. $\overline{BI}$ is the blanking input that shares the same pin with $\overline{RBO}$; in other words, the $\overline{BI}/\overline{RBO}$ pin can be used as an input or an output. When used as a $\overline{BI}$ (blanking input), all segment outputs are HIGH (nonactive) when $\overline{BI}$ is LOW, which overrides all other inputs. The $\overline{BI}$ function is not part of the zero suppression capability of the device.



(a) Illustration of leading zero suppression



(b) Illustration of trailing zero suppression

▲ **FIGURE 6–39**

All of the segment outputs of the decoder are nonactive (HIGH) if a zero code (0000) is on its BCD inputs and if its $\overline{RBI}$ is LOW. This causes the display to be blank and produces a LOW $\overline{RBO}$.

The logic diagram in Figure 6–39(a) illustrates leading zero suppression for a whole number. The highest-order digit position (left-most) is always blanked if a zero code is on its BCD inputs because the $\overline{RBI}$ of the most-significant decoder is made LOW by connecting it to ground. The $\overline{RBO}$ of each decoder is connected to the $\overline{RBI}$ of the next lowest-order decoder so that all zeros to the left of the first nonzero digit are blanked. For example, in part (a) of the figure the two highest-order digits are zeros and therefore are blanked. The remaining two digits, 3 and 9 are displayed.

The logic diagram in Figure 6–39(b) illustrates trailing zero suppression for a fractional number. The lowest-order digit (right-most) is always blanked if a zero code is on its BCD inputs because the $\overline{RBI}$ is connected to ground. The $\overline{RBO}$ of each decoder is connected to the $\overline{RBI}$ of the next highest-order decoder so that all zeros to the right of the first nonzero digit are blanked. In part (b) of the figure, the two lowest-order digits are zeros and therefore are blanked. The remaining two digits, 5 and 7 are displayed. To combine both leading and trailing zero suppression in one display and to have decimal point capability, additional logic is required.

SECTION 6–5 REVIEW

1. A 3-line-to-8-line decoder can be used for octal-to-decimal decoding. When a binary 101 is on the inputs, which output line is activated?
2. How many 74HC154 1-of-16 decoders are necessary to decode a 6-bit binary number?
3. Would you select a decoder/driver with active-HIGH or active-LOW outputs to drive a common-cathode 7-segment LED display?

6–6 ENCODERS

An encoder is a combinational logic circuit that essentially performs a "reverse" decoder function. An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. Encoders can also be devised to encode various symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called *encoding*.

After completing this section, you should be able to

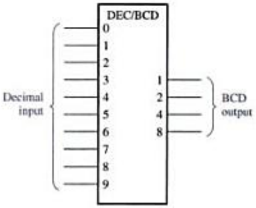
- Determine the logic for a decimal encoder
- Explain the purpose of the priority feature in encoders
- Describe the 74HC147 decimal-to-BCD priority encoder
- Describe the 74F148 octal-to-binary priority encoder
- Expand an encoder
- Apply the encoder to a specific application

The Decimal-to-BCD Encoder

This type of **encoder** has ten inputs—one for each decimal digit—and four outputs corresponding to the BCD code, as shown in Figure 6–40. This is a basic 10-line-to-4-line encoder.

FIGURE 6–40

Logic symbol for a decimal-to-BCD encoder.



The BCD (8421) code is listed in Table 6–6. From this table you can determine the relationship between each BCD bit and the decimal digits in order to analyze the logic. For instance, the most significant bit of the BCD code, A_3, is always a 1 for decimal digit 8 or 9. An OR expression for bit A_3 in terms of the decimal digits can therefore be written as

$$A_3 = 8 + 9$$

Bit A_2 is always a 1 for decimal digit 4, 5, 6 or 7 can be expressed as an OR function as follows:

$$A_2 = 4 + 5 + 6 + 7$$

Bit A_1 is always a 1 for decimal digit 2, 3, 6, or 7 and can be expressed as

$$A_1 = 2 + 3 + 6 + 7$$

Finally, A_0 is always a 1 for decimal digit 1, 3, 5, 7, or 9. The expression for A_0 is

$$A_0 = 1 + 3 + 5 + 7 + 9$$

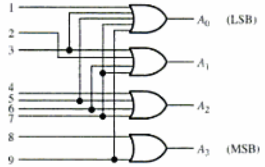
TABLE 6–6

	BCD CODE			
DECIMAL DIGIT	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Now, let us implement the logic circuitry required for encoding each decimal digit to a BCD code by using the logic expressions just developed. It is simply a matter of ORing the appropriate decimal digit input lines to form each BCD output. The basic encoder logic resulting from these expressions is shown in Figure 6–41.

The basic operation of the circuit in Figure 6–41 is as follows: When a HIGH appears on *one* of the decimal digit input lines, the appropriate levels occur on the four BCD output lines. For instance, if input line 9 is HIGH (assuming all other input lines are LOW), this condition

FIGURE 6–41


1
2
3
4
5
6
7
8
9
A_0 (LSB)
A_1
A_2
A_3 (MSB)

will produce a HIGH on outputs A_0 and A_3 and LOWs on outputs A_1 and A_2, which is the BCD code (1001) for decimal 9.

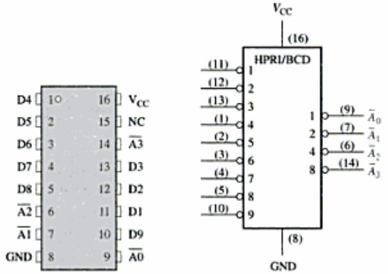
The Decimal-to-BCD Priority Encoder This type of encoder performs the same basic encoding function as previously discussed. It also offers additional flexibility in that it can be used in applications that require priority detection. The priority function means that the encoder will produce a BCD output corresponding to the *highest-order decimal digit* input that is active and will ignore any other lower-order active inputs. For instance, if the 6 and the 3 inputs are both active, the BCD output is 0110 (which represents decimal 6).

ENCODERS

A Decimal-to-BCD Encoder

The 74HC147 is a **priority encoder** with active-LOW inputs (0) for decimal digits 1 through 9 and active-LOW BCD outputs as indicated in the logic symbol in Figure 6–42. A BCD zero output is represented when none of the inputs is active. The device pin numbers are in parentheses.

FIGURE 6–42

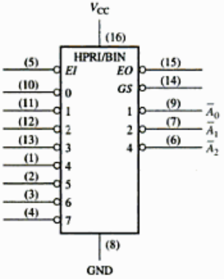

D4 1 16 V_{CC}
D5 2 15 NC
D6 3 14 $\overline{A3}$
D7 4 13 D3
D8 5 12 D2
$\overline{A2}$ 6 11 D1
$\overline{A1}$ 7 10 D9
GND 8 9 $\overline{A0}$
V_{CC}
(16)
HPRI/BCD
(11) 1
(12) 2
(13) 3
(1) 4
(2) 5
(3) 6
(4) 7
(5) 8
(10) 9
1 (9) $\overline{A}_0$
2 (7) $\overline{A}_1$
4 (6) $\overline{A}_2$
8 (14) $\overline{A}_3$
(8)
GND

(a) Pin diagram (b) Logic diagram

An 8-Line-to-3-Line Encoder

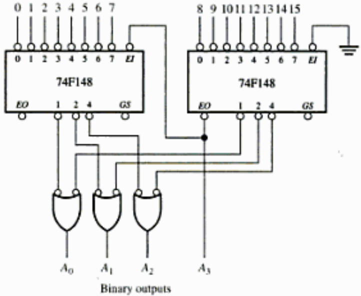
The 74F148 is a priority encoder that has eight active-LOW inputs and three active-LOW binary outputs, as shown in Figure 6–43. This device can be used for converting octal inputs (recall that the octal digits are 0 through 7) to a 3-bit binary code. To enable the device, the *EI* (enable input) must be LOW. It also has the *EO* (enable output) and *GS* output for expansion purposes. The *EO* is LOW when the *EI* is LOW and none of the inputs (0 through 7) is active. *GS* is LOW when *EI* is LOW and any of the inputs is active.

▶ **FIGURE 6–43**


V_{CC}
(16)
HPRI/BIN
(5)
EI
EO
(15)
(10)
0
GS
(14)
(11)
1
1
(9)
$\overline{A}_0$
(12)
2
2
(7)
$\overline{A}_1$
(13)
3
4
(6)
$\overline{A}_2$
(1)
4
(2)
5
(3)
6
(4)
7
(8)
GND

The 74F148 can be expanded to a 16-line-to-4-line encoder by connecting the *EO* of the higher-order encoder to the *EI* of the lower-order encoder and negative-ORing the corresponding binary outputs as shown in Figure 6–44. The *EO* is used as the fourth and most-significant bit. This particular configuration produces active-HIGH outputs for the 4-bit binary number.

▶ **FIGURE 6–44**


0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 EI
74F148
EO 1 2 4 GS
0 1 2 3 4 5 6 7 EI
74F148
EO 1 2 4 GS
A_0
A_1
A_2
A_3
Binary outputs

EXAMPLE 6–11

If LOW levels appear on pins, 1, 4, and 13 of the 74HC147 shown in Figure 6–42, indicate the state of the four outputs. All other inputs are HIGH.

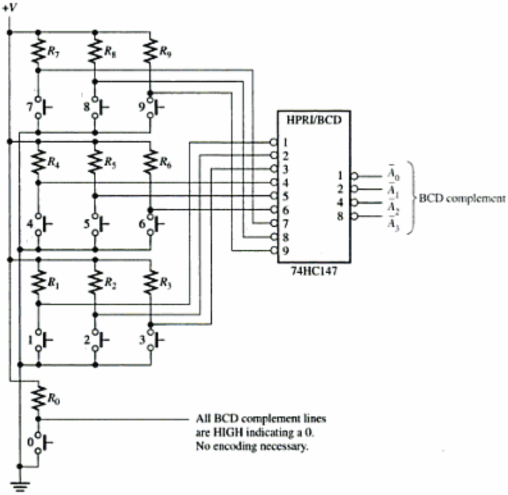
Solution Pin 4 is the highest-order decimal digit input having a LOW level and represents decimal 7. Therefore, the output levels indicate the BCD code for decimal 7 where $\overline{A}_0$ is the LSB and $\overline{A}_3$ is the MSB. Output $\overline{A}_0$ is LOW, $\overline{A}_1$ is LOW, $\overline{A}_2$ is LOW, and $\overline{A}_3$ is HIGH.

Supplementary Problem What are the outputs of the 74HC147 if all its inputs are LOW? If all its inputs are HIGH?

Application Example

A classic application example is a keyboard encoder. The ten decimal digits on the keyboard of a computer, for example, must be encoded for processing by the logic circuitry. When one of the keys is pressed, the decimal digit is encoded to the corresponding BCD code. Figure 6–45 shows a simple keyboard encoder arrangement using a 74HC147 priority encoder. The keys are represented by ten push-button switches, each with a **pull-up resistor** to $+V$. The pull-up resistor ensures that the line is HIGH when a key is not depressed. When a key is depressed, the line is connected to ground, and a LOW is applied to the corresponding encoder input. The zero key is not connected because the BCD output represents zero when none of the other keys is depressed.

▶ **FIGURE 6–45**



All BCD complement lines are HIGH indicating a 0. No encoding necessary.

The BCD complement output of the encoder goes into a storage device, and each successive BCD code is stored until the entire number has been entered. Methods of storing BCD numbers and binary data are covered in later chapters.

SECTION 6–6 REVIEW

1. Suppose, the HIGH levels are applied to the 2 input and the 9 input of the circuit in Figure 6–41.
   (a) What are the states of the output lines?
   (b) Does this represent a valid BCD code?
   (c) What is the restriction on the encoder logic in Figure 6–41?
2. (a) What is the $\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$ output when LOWs are applied to pins 1 and 5 of the 74HC147 in Figure 6–42?
   (b) What does this output represent?

6–7 CODE CONVERTERS

In this section, we will examine some methods of using combinational logic circuits to convert from one code to another.

After completing this section, you should be able to

- Explain the process for converting BCD to binary
- Use exclusive-OR gates for conversions between binary and Gray codes

BCD-to-Binary Conversion

One method of BCD-to-binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number is represented by a binary number.
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added.
3. The result of this addition is the binary equivalent of the BCD number.

A more concise statement of this operation is

The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

Let us examine an 8-bit BCD code (one that represents a 2-digit decimal number) to understand the relationship between BCD and binary. For instance, you already know that the decimal number 87 can be expressed in BCD as

$$\underbrace{1000}_{8}\quad\underbrace{0111}_{7}$$

The left-most 4-bit group represents 80, and the right-most 4-bit group represents 7. That is, the left-most group has a weight of 10, and the right-most group has a weight of 1. Within each group, the binary weight of each bit is as follows:

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This representation is given in Table 6–7.

TABLE 6–7

		(MSB)	BINARY REPRESENTATION					(LSB)
BCD BIT	BCD WEIGHT	64	32	16	8	4	2	1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

If the binary representations for the weights of all the 1s in the BCD number are added, the result is the binary number that corresponds to the BCD number. Example 6–12 illustrates this.

EXAMPLE 6–12

Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Solution Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

80 40 20 10 8 4 2 1
0 0 1 0 0 1 1 1

0000001	1
0000010	2
0000100	4
+ 0010100	20
0011011	Binary number for decimal 27

80 40 20 10 8 4 2 1
1 0 0 1 1 0 0 0

0001000	8
0001010	10
+ 1010000	80
1100010	Binary number for decimal 98

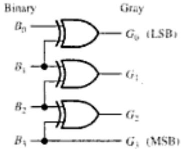
Supplementary Problem Show the process of converting 01000001 in BCD to binary.

With this basic procedure in mind, let us see how the process can be implemented with logic circuits. Once the binary representation for each 1 in the BCD number is determined, adder circuits can be used to add the 1s in each column of the binary representation. The 1s occur in a given column only when the corresponding BCD bit is a 1. The occurrence of a BCD 1 can therefore be used to generate the proper binary 1 in the appropriate column of the adder structure. To handle a two-decimal-digit (two-decade) BCD code, eight BCD input lines and seven binary outputs are required. (It takes seven bits to represent binary numbers through ninety-nine.)

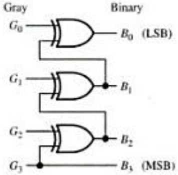
Binary-to-Gray and Gray-to-Binary Conversion

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions. Programmable logic devices (PLDs) can also be programmed for these code conversions. Figure 6–46 shows a 4-bit binary-to-Gray code converter, and Figure 6–47 illustrates a 4-bit Gray-to-binary converter.

► **FIGURE 6–46**



▶ **FIGURE 6–47**



EXAMPLE 6–13

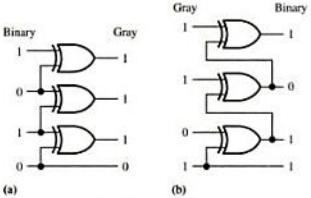
(a) Convert the binary number 0101 to Gray code with exclusive-OR gates.

(b) Convert the Gray code 1011 to binary with exclusive-OR gates.

Solution **(a) 0101_2 is 0111 Gray. See Figure 6–48(a).**

(b) 1011 Gray is 1101_2. See Figure 6–48(b).

▶ **FIGURE 6–48**



Supplementary Problem **How many exclusive-OR gates are required to convert 8-bit binary to Gray?**

SECTION 6–7 REVIEW

1. Convert the BCD number 10000101 to binary.
2. Draw the logic diagram for converting an 8-bit binary number to Gray code.

6–8 MULTIPLEXERS (DATA SELECTORS)

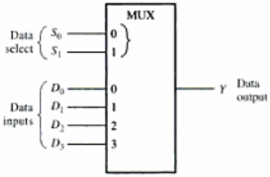
A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination. The basic multiplexer has several data-input lines and a single output line. It also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line. Multiplexers are also known as data selectors.

After completing this section, you should be able to

■ Explain the basic operation of a multiplexer ■ Describe the 74LS151 and the 74HC157A multiplexers ■ Expand a multiplexer to handle more data inputs ■ Use the multiplexer as a logic function generator

A logic symbol for a 4-input **multiplexer (MUX)** is shown in Figure 6–49. Notice that there are two data-select lines because with two select bits, any one of the four data-input lines can be selected.

▶ **FIGURE 6–49**



In Figure 6–49, a 2-bit code on the data-select (S) inputs will allow the data on the selected data input to pass through to the data output. If a binary 0 ($S_1 = 0$ and $S_0 = 0$) is applied to the data-select lines, the data on input D_0 appear on the data-output line. If a binary 1 ($S_1 = 0$ and $S_0 = 1$) is applied to the data-select lines, the data on input D_1 appear on the data output. If a binary 2 ($S_1 = 1$ and $S_0 = 0$) is applied, the data on D_2 appear on the output. If a binary 3 ($S_1 = 1$ and $S_0 = 1$) is applied, the data on D_3 are switched to the output line. A summary of this operation is given in Table 6–8.

▶ **TABLE 6–8**

DATA-SELECT INPUTS		INPUT SELECTED
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Now, let us look at the logic circuitry required to perform this multiplexing operation. The data output is equal to the state of the *selected* data input. You can therefore, derive a logic expression for the output in terms of the data input and the select inputs.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.

The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1S_0$.

The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1\overline{S}_0$.

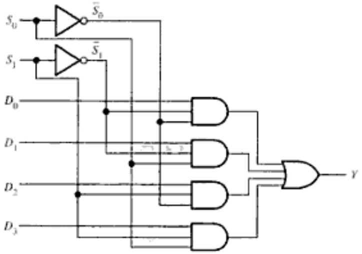
The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

When these terms are ORed, the total expression for the data output is

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

The implementation of this equation requires four 3-input AND gates, a 4-input OR gate, and two inverters to generate the complements of S_1 and S_0, as shown in Figure 6–50. Because data can be selected from any one of the input lines, this circuit is also referred to as a **data selector**.

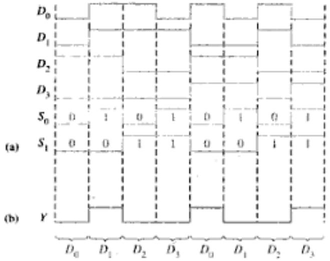
FIGURE 6–50



EXAMPLE 6–14

The data-input and data-select waveforms in Figure 6–51(a) are applied to the multiplexer in Figure 6–50. Determine the output waveform in relation to the inputs.

FIGURE 6–51



Solution The binary state of the data-select inputs during each interval determines which data input is selected. Notice that the data-select inputs go through a repetitive binary sequence 00, 01, 10, 11, 00, 01, 10, 11, and so on. The resulting output waveform is shown in Figure 6–51(b).

Supplementary Problem Construct a timing diagram showing all inputs and the output if the S_0 and S_1 waveforms in Figure 6–51(a) are interchanged.

MULTIPLEXERS

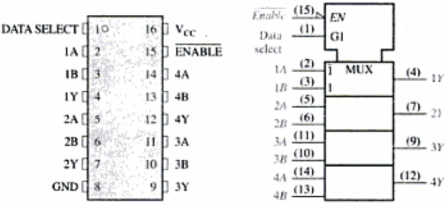
A Quad 2-Input Data Selector/Multiplexer

The 74HC157A, as well as its LS version, consists of four separate 2-input multiplexers. Each of the four multiplexers shares a common data-select line and a common *Enable*. Because there are only two inputs to be selected in each multiplexer, a single data-select input is sufficient.

A LOW on the $\overline{Enable}$ input allows the selected input data to pass through to the output. A HIGH on the $\overline{Enable}$ input prevents data from going through to the output; that is, it disables the multiplexers.

The ANSI/IEEE Logic Symbol The pin diagram for the 74HC157A is shown in Figure 6–52(a). The ANSI/IEEE logic symbol for the 74HC157A is shown in Figure 6–52(b). Notice that the four multiplexers are indicated by the partitioned outline and that the inputs common to all four multiplexers are indicated as inputs to the notched block at the top, which is called the *common control block*. All labels within the upper MUX block apply to the other blocks below it.

▶ FIGURE 6–52



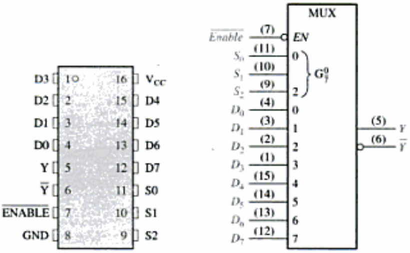
(a) Pin diagram (b) Logic symbol

Notice the 1 and $\overline{1}$ labels in the MUX blocks and the G1 label in the common control block. These labels are an example of the **dependency notation** system specified in the ANSI/IEEE Standard 91–1984. In this case G1 indicates an AND relationship between the data-select input and the data inputs with 1 or $\overline{1}$ labels. (The $\overline{1}$ means that the AND relationship applies to the complement of the G1 input.) In other words, when the data-select input is HIGH, the *B* inputs of the multiplexers are selected; and when the data-select input is LOW, the *A* inputs are selected. A "G" is always used to denote AND dependency. Other aspects of dependency notation are introduced as appropriate throughout the book.

An 8-Input Data Selector/Multiplexer

The 74LS151 has eight data inputs (D_0–D_7) and, therefore, three data-select or address input lines (S_0–S_2). Three bits are required to select any one of the eight data inputs ($2^3 = 8$). A LOW on the $\overline{Enable}$ input allows the selected input data to pass through to the output. Notice that the data output and its complement are both available. The pin diagram is shown in Figure 6–53(a), and the ANSI/IEEE logic symbol is shown in part (b). In this case there is no need for a common control block on the logic symbol because there is only one multiplexer to be

▶ FIGURE 6–53



(a) Pin diagram (b) Logic symbol

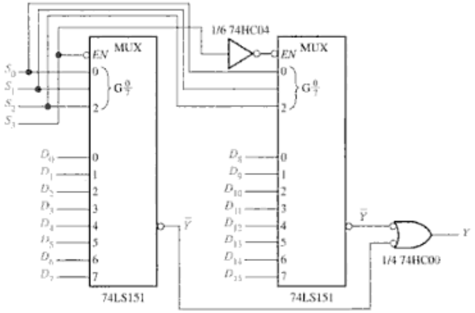
controlled, not four as in the 74HC157A. The G_7^0 label within the logic symbol indicates the AND relationship between the data-select inputs and each of the data inputs 0 through 7.

EXAMPLE 6–15

Use 74LS151s and any other logic necessary to multiplex 16 data lines onto a single data-output line.

Solution An implementation of this system is shown in Figure 6–54. Four bits are required to select one of 16 data inputs ($2^4 = 16$). In this application the $\overline{Enable}$ input is used as the most significant data-select bit. When the MSB in the data-select code is LOW, the left 74LS151 is enabled, and one of the data inputs (D_0 through D_7) is selected by the other three data-select bits. When the data-select MSB is HIGH, the right 74LS151 is enabled, and one of the data inputs (D_8 through D_{15}) is selected. The selected input data are then passed through to the negative-OR gate and onto the single output line.

FIGURE 6–54



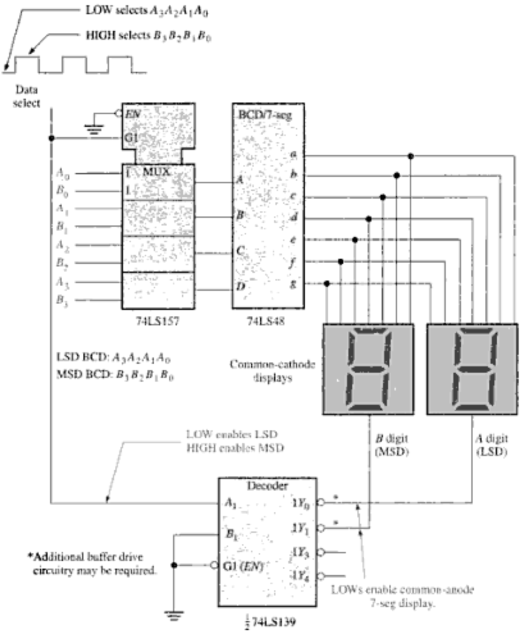
Supplementary Problem Determine the codes on the select inputs required to select each of the following data inputs: D_0, D_4, D_8, and D_{13}.

Application Examples

A 7-Segment Display Multiplexer Figure 6–55 shows a simplified method of multiplexing BCD numbers to a 7-segment display. In this example, 2-digit numbers are displayed on the 7-segment readout by the use of a single BCD-to-7-segment decoder. This basic method of display multiplexing can be extended to displays with any number of digits. The basic operation is as follows.

Two BCD digits ($A_3A_2A_1A_0$ and $B_3B_2B_1B_0$) are applied to the multiplexer inputs. A square wave is applied to the data-select line, and when it is LOW, the A bits ($A_3A_2A_1A_0$) are passed through to the inputs of the 74LS48 BCD-to-7-segment decoder. The LOW on the data-select also puts a LOW on the A_1 input of the 74LS139 2-line-to-4-line decoder, thus activating its 0 output and enabling the A-digit display by effectively connecting its common terminal to ground. The A digit is now *on* and the B digit is *off*.

When the data-select line goes HIGH, the B bits ($B_3B_2B_1B_0$) are passed through to the inputs of the BCD-to-7-segment decoder. Also, the 74LS139 decoder's 1 output is activated, thus enabling the B-digit display. The B digit is now *on* and the A digit is *off*. The cycle repeats at the frequency of the data-select square wave. This frequency must be high enough (about 30 Hz) to prevent visual flicker as the digit displays are multiplexed.


LOW selects $A_3A_2A_1A_0$
HIGH selects $B_3B_2B_1B_0$
Data select
EN
G1
MUX
A_0
B_0
A_1
B_1
A_2
B_2
A_3
B_3
74LS157
BCD/7-seg
A
B
C
D
a
b
c
d
e
f
g
74LS48
LSD BCD: $A_3A_2A_1A_0$
MSD BCD: $B_3B_2B_1B_0$
Common-cathode displays
LOW enables LSD
HIGH enables MSD
B digit (MSD)
A digit (LSD)
Decoder
A_1
B_1
G1 (EN)
$1Y_0$
$1Y_1$
$1Y_3$
$1Y_4$
*Additional buffer drive circuitry may be required.
LOWs enable common-anode 7-seg display.
$\frac{1}{2}$ 74LS139

▲ **FIGURE 6–55**

A Logic Function Generator A useful application of the data selector/multiplexer is in the generation of combinational logic functions in sum-of-products form. When used in this way, the device can replace discrete gates, can often greatly reduce the number of ICs, and can make design changes much easier.

To illustrate, a 74LS151 8-input data selector/multiplexer can be used to implement any specified 3-variable logic function if the variables are connected to the data-select inputs and each data input is set to the logic level required in the truth table for that function. For example, if the function is a 1 when the variable combination is $\overline{A}_2A_1\overline{A}_0$, the 2 input (selected by 010) is connected to a HIGH. This HIGH is passed through to the output when this particular combination of variables occurs on the data-select lines. An example will help clarify this application.

EXAMPLE 6–16

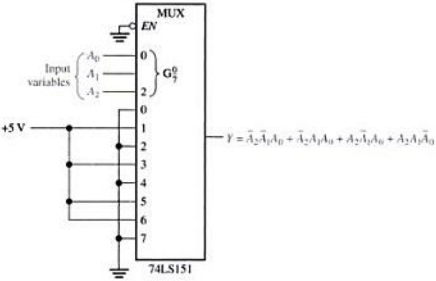
Implement the logic function specified in Table 6–9 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

▶ TABLE 6–9

INPUTS			OUTPUT
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Solution Notice from the truth table that Y is a 1 for the following input variable combinations: 001, 011, 101, and 110. For all other combinations, Y is 0. For this function to be implemented with the data selector, the data input selected by each of the above-mentioned combinations must be connected to a HIGH (5 V). All the other data inputs must be connected to a LOW (ground), as shown in Figure 6–56.

▶ FIGURE 6–56


MUX
EN
Input variables
A_0
A_1
A_2
0
2
$G\frac{0}{7}$
+5 V
0
1
2
3
4
5
6
7
$Y = \overline{A}_2\overline{A}_1A_0 + \overline{A}_2A_1A_0 + A_2\overline{A}_1A_0 + A_2A_1\overline{A}_0$
74LS151

The implementation of this function with logic gates would require four 3-input AND gates, one 4-input OR gate, and three inverters unless the expression can be simplified.

Supplementary Problem Use the 74LS151 to implement the following expression:

$$Y = \overline{A}_2\overline{A}_1\overline{A}_0 + A_2\overline{A}_1\overline{A}_0 + \overline{A}_2A_1\overline{A}_0$$

Example 6–16 illustrated how the 8-input data selector can be used as a logic function generator for three variables. Actually, this device can be also used as a 4-variable logic function generator by the utilization of one of the bits (A_0) in conjunction with the data inputs.

A 4-variable truth table has sixteen combinations of input variables. When an 8-bit data selector is used, each input is selected twice: the first time when A_0 is 0 and the second time when A_0 is 1. With this in mind, the following rules can be applied (Y is the output, and A_0 is the least significant bit):

1. If $Y = 0$ both times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, connect that data input to ground (0).
2. If $Y = 1$ both times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, connect the data input to $+V(1)$.
3. If Y is different the two times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, and if $Y = A_0$, connect that data input to A_0.
4. If Y is different the two times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, and if $Y = \overline{A}_0$, connect that data input to $\overline{A}_0$.

The following example illustrates this method.

EXAMPLE 6–17

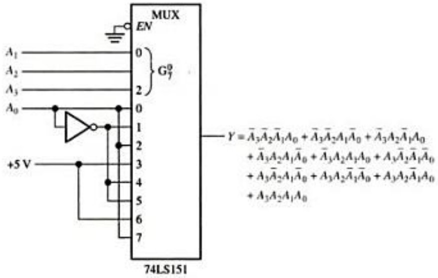
Implement the logic function in Table 6–10 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

► **TABLE 6–10**

DECIMAL DIGIT	INPUTS				OUTPUT
	A_3	A_2	A_1	A_0	Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Solution The data-select inputs are $A_3A_2A_1$. In the first row of the table, $A_3A_2A_1 = 000$ and $Y = A_0$. In the second row, where $A_3A_2A_1$ again is 000, $Y = A_0$. Thus, A_0 is connected to the 0 input. In the third row of the table, $A_3A_2A_1 = 001$ and $Y = \overline{A}_0$. Also, in the fourth row, when $A_3A_2A_1$ again is 001, $Y = \overline{A}_0$. Thus, A_0 is inverted and connected to the 1 input. This analysis is continued until each input is properly connected according to the specified rules. The implementation is shown in Figure 6–57.

▶ FIGURE 6–57


MUX
EN
A_1
A_2
A_3
A_0
+5 V
0
2
$G\frac{0}{7}$
0
1
2
3
4
5
6
7
$Y = \overline{A}_3\overline{A}_2\overline{A}_1A_0 + \overline{A}_3\overline{A}_2A_1\overline{A}_0 + \overline{A}_3A_2\overline{A}_1A_0$
$+ \overline{A}_3A_2A_1\overline{A}_0 + \overline{A}_3A_2A_1A_0 + A_3\overline{A}_2\overline{A}_1\overline{A}_0$
$+ A_3\overline{A}_2A_1\overline{A}_0 + A_3A_2\overline{A}_1\overline{A}_0 + A_3A_2\overline{A}_1A_0$
$+ A_3A_2A_1A_0$
74LS151

If implemented with logic gates, the function would require as many as ten 4-input AND gates, one 10-input OR gate, and four inverters, although possible simplification would reduce this requirement.

Supplementary Problem In Table 6–10, if $Y = 0$ when the inputs are all zeros and is alternately a 1 and a 0 for the remaining rows in the table, use a 74LS151 to implement the resulting logic function.

SECTION 6–8 REVIEW

1. In Figure 6–50, $D_0 = 1$, $D_1 = 0$, $D_2 = 1$, $D_3 = 0$, $S_0 = 1$, and $S_1 = 0$. What is the output?
2. Identify each device:
   (a) 74LS157 (b) 74LS151
3. A 74LS151 has alternating LOW and HIGH levels on its data inputs beginning with $D_0 = 0$. The data-select lines are sequenced through a binary count (000, 001, 010, and so on) at a frequency of 1 kHz. The enable input is LOW. Describe the data output waveform.
4. Briefly describe the purpose of each of the following devices in Figure 6–55:
   (a) 74LS157 (b) 74LS48 (c) 74LS139

6–9 DEMULTIPLEXERS

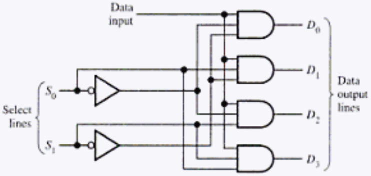
A demultiplexer (DEMUX) basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. As you will learn, decoders can also be used as demultiplexers.

After completing this section, you should be able to

- Explain the basic operation of a demultiplexer
- Describe how the 74HC154 4-line-to-16-line decoder can be used as a demultiplexer
- Develop the timing diagram for a demultiplexer with specified data and data selection inputs

Figure 6–58 shows a 1-line-to-4-line **demultiplexer (DEMUX)** circuit. The data-input line goes to all of the AND gates. The two data-select lines enable only one gate at a time, and the data appearing on the data-input line will pass through the selected gate to the associated data-output line.

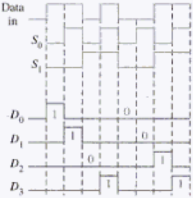
► **FIGURE 6–58**



EXAMPLE 6–18

The serial data-input waveform (Data in) and data-select inputs (S_0 and S_1) are shown in Figure 6–59. Determine the data-output waveforms on D_0 through D_3 for the demultiplexer in Figure 6–58.

► **FIGURE 6–59**



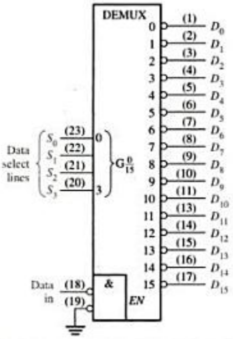
Solution Notice that the select lines go through a binary sequence so that each successive input bit is routed to D_0, D_1, D_2, and D_3 in sequence, as shown by the output waveforms in Figure 6–59.

Supplementary Problem Develop the timing diagram for the demultiplexer if the S_0 and S_1 waveforms are both inverted.

A DEMULTIPLEXER

We have already discussed the 74HC154 decoder in its application as a 4-line-to-16-line decoder (Section 6–5). This device and other decoders can also be used in demultiplexing applications. The logic symbol for this device when used as a demultiplexer is shown in Figure 6–60. In demultiplexer applications, the input lines are used as the data-select lines. One of the chip select inputs is used as the data-input line, with the other chip select input held LOW to enable the internal negative-AND gate at the bottom of the diagram.

FIGURE 6–60



SECTION 6–9 REVIEW

1. Generally, how can an decoder be used as a demultiplexer?
2. The 74HC154 demultiplexer in Figure 6–60 has a binary code of 1010 on the data-select lines, and the data-input line is LOW. What are the states of the output lines?

6–10 PARITY GENERATORS/CHECKERS

Errors can occur as digital codes are being transferred from one point to another within a digital system or while codes are being transmitted from one system to another. The errors take the form of undesired changes in the bits that make up the coded information; that is, a 1 can change to a 0, or a 0 to a 1, because of component malfunctions or electrical noise. In most digital systems, the probability that even a single bit error will occur is very small, and the likelihood that more than one will occur is even smaller. Nevertheless, when an error occurs undetected, it can cause serious problems in a digital system.

After completing this section, you should be able to

- Explain the concept of parity
- Implement a basic parity circuit with exclusive-OR gates
- Describe the operation of basic parity generating and checking logic
- Discuss the 74LS280 9-bit parity generator/checker
- Discuss how error detection can be implemented in a data transmission

The parity method of error detection in which a **parity bit** is attached to a group of information bits in order to make the total number of 1s either even or odd (depending on the system) was covered in Chapter 2. In addition to parity bits, several specific codes also provide inherent error detection.

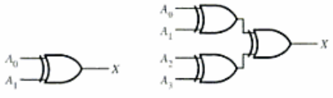
Basic Parity Logic

In order to check for or to generate the proper parity in a given code, a basic principle can be used:

> **The sum (disregarding carries) of an even number of 1s is always 0, and the sum of an odd number of 1s is always 1.**

Therefore, to determine if a given code has **even parity** or **odd parity**, all the bits in that code are summed. As you know, the sum of two bits can be generated by an exclusive-OR gate, as shown in Figure 6–61(a); the sum of four bits can be formed by three exclusive-OR gates connected as shown in Figure 6–61(b); and so on. When the number of 1s on the inputs is even, the output X is 0 (LOW). When the number of 1s is odd, the output X is 1 (HIGH).

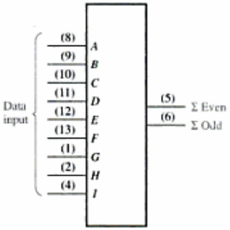
▶ **FIGURE 6–61**



(a) Summing of two bits (b) Summing of four bits

A 9-BIT PARITY GENERATOR/CHECKER

The logic symbol and function table for a 74LS280 are shown in Figure 6–62. This particular device can be used to check for odd or even parity on a 9-bit code (eight data bits and one parity bit) or it can be used to generate a parity bit for a binary code with up to nine bits. The inputs are A through I; when there is an even number of 1s on the inputs, the Σ Even output is HIGH and the Σ Odd output is LOW.



(a) Traditional logic symbol

Number of Inputs A–I That Are HIGH	Outputs Σ Even	Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

(b) Function table

▲ **FIGURE 6–62**

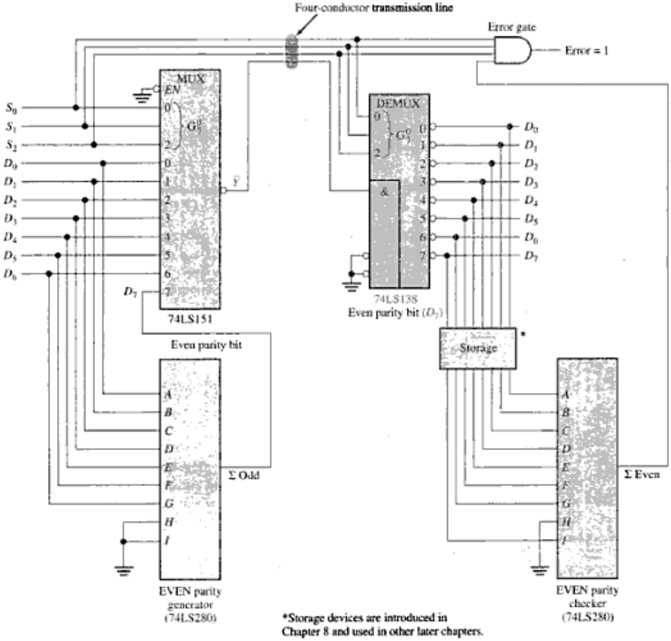
Parity Checker When this device is used as an even parity checker, the number of input bits should always be even; and when a parity error occurs, the Σ Even output goes LOW and the

Σ Odd output goes HIGH. When it is used as an odd parity checker, the number of input bits should always be odd; and when a parity error occurs, the Σ Odd output goes LOW and the Σ Even output goes HIGH.

Parity Generator If this device is used as an even parity generator, the parity bit is taken at the Σ Odd output because this output is a 0 if there is an even number of input bits and it is a 1 if there is an odd number. When used as an odd parity generator, the parity bit is taken at the Σ Even output because it is a 0 when the number of inputs bits is odd.

A Data Transmission System with Error Detection

A simplified data transmission system is shown in Figure 6–63 to illustrate an application of parity generators/checkers, as well as multiplexers and demultiplexers, and to illustrate the need for data storage in some applications.


Four-conductor transmission line
Error gate
Error = 1
MUX
EN
$G\frac{0}{7}$
S_0
S_1
S_2
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$\bar{Y}$
74LS151
Even parity bit
DEMUX
$G\frac{0}{7}$
&
74LS138
Even parity bit (D_7)
Storage
A
B
C
D
E
F
G
H
I
Σ Odd
Σ Even
EVEN parity generator (74LS280)
EVEN parity checker (74LS280)

*Storage devices are introduced in Chapter 8 and used in other later chapters.

▲ **FIGURE 6–63**

In this application, digital data from seven sources are multiplexed onto a single line for transmission to a distant point. The seven data bits (D_0 through D_6) are applied to the multiplexer data inputs and, at the same time, to the even parity generator inputs. The Σ Odd output of the parity generator is used as the even parity bit. This bit is 0 if the number of 1s on the inputs A through I is even and is a 1 if the number of 1s on A through I is odd. This bit is D_7 of the transmitted code.

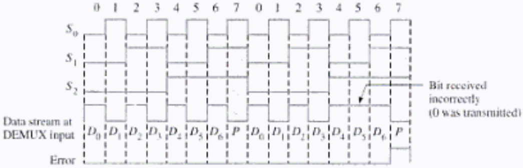
The data-select inputs are repeatedly cycled through a binary sequence, and each data bit, beginning with D_0, is serially passed through and onto the transmission line ($\overline{Y}$). In this example, the transmission line consists of four conductors: one carries the serial data and three carry the timing signals (data selects). There are more sophisticated ways of sending the timing information, but we are using this direct method to illustrate a basic principle.

At the demultiplexer end of the system, the data-select signals and the serial data stream are applied to the demultiplexer. The data bits are distributed by the demultiplexer onto the output lines in the order in which they occurred on the multiplexer inputs. That is, D_0 comes out on the D_0 output, D_1 comes out on the D_1 output, and so on. The parity bit comes out on the D_7 output. These eight bits are temporarily stored and applied to the even parity checker. Not all of the bits are present on the parity checker inputs until the parity bit D_7 comes out and is stored. At this time, the error gate is enabled by the data-select code 111. If the parity is correct, a 0 appears on the Σ Even output, keeping the Error output at 0. If the parity is incorrect, all 1s appear on the error gate inputs, and a 1 on the Error output results.

This particular application has demonstrated the need for data storage so that you will be better able to appreciate the usefulness of the storage devices introduced in Chapter 7 and used in other later chapters.

The timing diagram in Figure 6–64 illustrates a specific case in which two 8-bit words are transmitted, one with correct parity and one with an error.

FIGURE 6–64


0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
S_0
S_1
S_2
Data stream at DEMUX input
D_0 D_1 D_2 D_3 D_4 D_5 D_6 P D_0 D_1 D_2 D_3 D_4 D_5 D_6 P
Error
Bit received incorrectly (0 was transmitted)

SECTION 6–10 REVIEW

1. Add an even parity bit to each of the following codes:
   (a) 110100 (b) 01100011.
2. Add an odd parity bit to each of the following codes:
   (a) 1010101 (b) 1000001.
3. Check each of the even parity codes for an error.
   (a) 100010101 (b) 1110111001.

6–11 GLITCHES IN DECODER CIRCUITS

In this section, the problem of decoder glitches is introduced. A glitch is any undesired voltage or current spike (pulse) of very short duration. A glitch can be interpreted as a valid signal by a logic circuit and may cause improper operation.

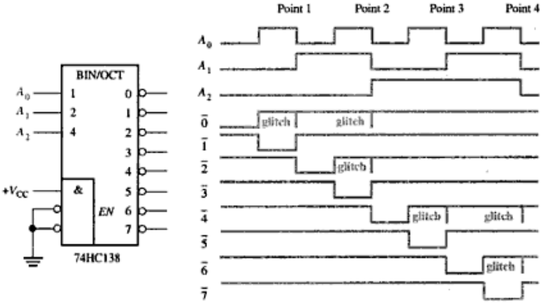
After completing this section, you should be able to

- Explain what a glitch is
- Determine the cause of glitches in a decoder application
- Use the method of output strobing to eliminate glitches

The 74LS138 was used as a DEMUX in the data transmission system in Figure 6–63. Now, the 74HC138 is used as a 3-line-to-8-line decoder (binary-to-octal) in Figure 6–65 to illustrate how **glitches** occur and how to identify their cause. The $A_2A_1A_0$ inputs of the decoder are sequenced through a binary count, and the resulting waveforms of the inputs and outputs are as shown in Figure 6–65. A_2 transitions are delayed from A_1 transitions and A_1 transitions are delayed from A_0 transitions. This commonly occurs when waveforms are generated by a binary counter, as you will learn in Chapter 8.

FIGURE 6–65

Decoder waveforms with output glitches


Point 1
Point 2
Point 3
Point 4
BIN/OCT
A_0
A_1
A_2
1
2
4
0
1
2
3
4
5
6
7
&
EN
+V_{CC}
74HC138
$\overline{0}$
$\overline{1}$
$\overline{2}$
$\overline{3}$
$\overline{4}$
$\overline{5}$
$\overline{6}$
$\overline{7}$
glitch
glitch
glitch
glitch
glitch
glitch

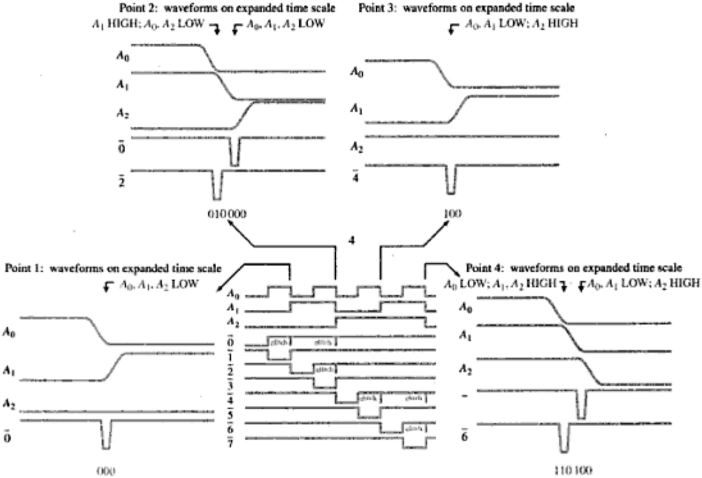

Point 2: waveforms on expanded time scale
A_1 HIGH; A_0, A_2 LOW
A_0, A_1, A_2 LOW
A_0
A_1
A_2
$\overline{0}$
$\overline{2}$
010 000
Point 3: waveforms on expanded time scale
A_0, A_1 LOW; A_2 HIGH
A_0
A_1
A_2
$\overline{4}$
100
4
Point 1: waveforms on expanded time scale
A_0, A_1, A_2 LOW
A_0
A_1
A_2
$\overline{0}$
000
Point 4: waveforms on expanded time scale
A_0 LOW; A_1, A_2 HIGH
A_0, A_1 LOW; A_2 HIGH
A_0
A_1
A_2
$\overline{6}$
110 100

FIGURE 6–66

Portions of waveforms on expanded time scale

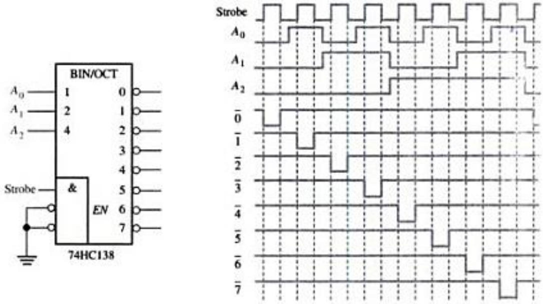
The output waveforms are correct except for the glitches that occur on some of the output signals. The points of interest are indicated on the input waveforms in Figure 6–65 and are shown in expanded form in Figure 6–66.

At point 1 there is a transitional state of 000 due to delay differences in the waveforms. This causes the first glitch on the $\overline{0}$ output of the decoder. At point 2 there are two transitional states, 010 and 000. These cause the glitch on the $\overline{2}$ output of the decoder and the second glitch on the $\overline{0}$ output, respectively. At point 3 the transitional state is 100, which causes the first glitch on the $\overline{4}$ output of the decoder. At point 4 the two transitional states, 110 and 100, result in the glitch on the $\overline{6}$ output and the second glitch on the $\overline{4}$ output, respectively.

One way to eliminate the glitch problem is a method called **strobing**, in which the decoder is enabled by a strobe pulse only during the times when the waveforms are not in transition. This method is illustrated in Figure 6–67.

▶ FIGURE 6–67

Use of strobe waveform to eliminate glitches



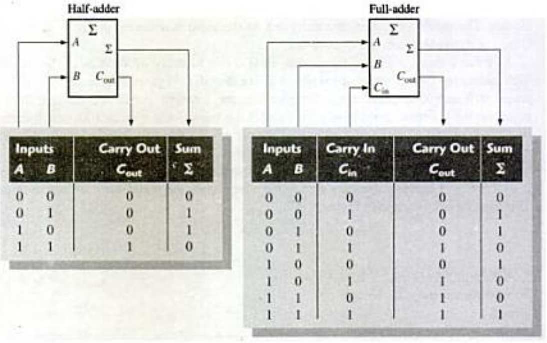
SECTION 6–11 REVIEW

1. Define the term *glitch*.
2. Explain the basic cause of glitches in decoder logic.
3. Define the term *strobe*.

SUMMARY

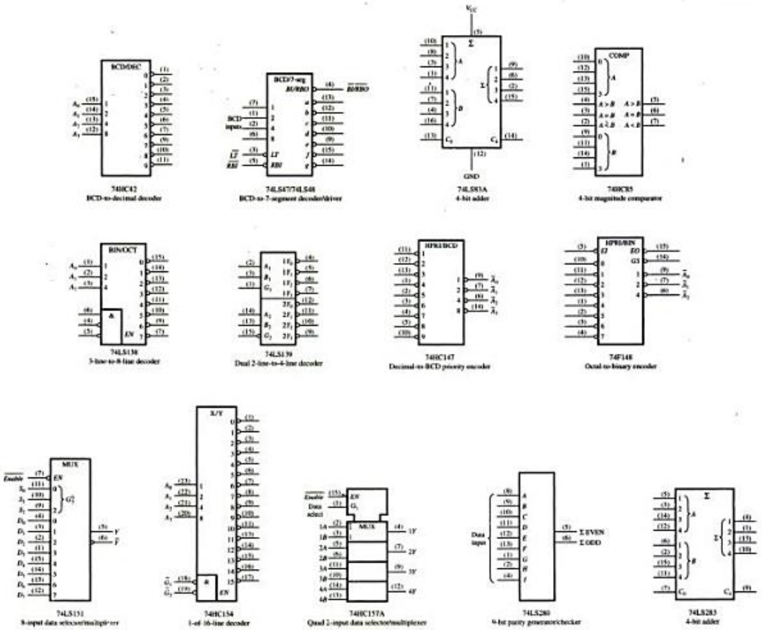
- The basic logic functions are comparison, arithmetic, code conversion, decoding, encoding, data selection, storage, and counting.
- Half-adder and full-adder operations are summarized in Figure 6–68.
- Logic symbols with pin numbers for the ICs used in this chapter are shown in Figure 6–69. Pin designations may differ from some manufacturers' data sheets.

FIGURE 6–68


Half-adder
Σ
A
B
Σ
C_{out}
Full-adder
Σ
A
B
C_{in}
Σ
C_{out}

Inputs A	Inputs B	Carry Out C_{out}	Sum Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Inputs A	Inputs B	Carry In C_{in}	Carry Out C_{out}	Sum Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1


BCD/DEC
BCD/7-seg
BI/RBO
BCD inputs
LT
RBI
Σ
V_{CC}
GND
C_0
C_4
COMP
A > B
A = B
A < B
BIN/OCT
&
EN
HPRI/BCD
HPRI/BIN
EI
EO
GS
MUX
Enable
EN
X/Y
Data select
Data input
Σ EVEN
Σ ODD

74HC42
BCD-to-decimal decoder

74LS47/74LS48
BCD-to-7-segment decoder/driver

74LS83A
4-bit adder

74HC85
4-bit magnitude comparator

74LS138
3-line-to-8-line decoder

74LS139
Dual 2-line-to-4-line decoder

74HC147
Decimal-to-BCD priority encoder

74F148
Octal-to-binary encoder

74LS151
8-input data selector/multiplexer

74HC154
1-of-16-line decoder

74HC157A
Quad 2-input data selector/multiplexer

74LS280
9-bit parity generator/checker

74LS283
4-bit adder

FIGURE 6–69

SELF-TEST

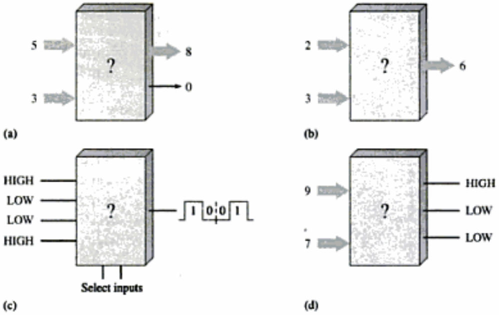
Answers are at the end of the chapter.

1. The device used to convert a binary number to a 7-segment display format is
   (a) multiplexer **(b)** encoder **(c)** decoder **(d)** register
2. An example of a data storage device is
   (a) the logic gate **(b)** the flip-flop **(c)** the comparator
   (d) the register **(e)** both answers (b) and (d)
3. A half-adder is characterized by
   (a) two inputs and two outputs **(b)** three inputs and two outputs
   (c) two inputs and three outputs **(d)** two inputs and one output
4. A full-adder is characterized by
   (a) two inputs and two outputs **(b)** three inputs and two outputs
   (c) two inputs and three outputs **(d)** two inputs and one output
5. The inputs to a full-adder are $A = 1, B = 1, C_{in} = 0$. The outputs are
   (a) $\Sigma = 1, C_{out} = 1$ **(b)** $\Sigma = 1, C_{out} = 0$
   (c) $\Sigma = 0, C_{out} = 1$ **(d)** $\Sigma = 0, C_{out} = 0$
6. A 4-bit parallel adder can add
   (a) two 4-bit binary numbers **(b)** two 2-bit binary numbers
   (c) four bits at a time **(d)** four bits in sequence
7. The 74LS83A is an example of a 4-bit parallel adder. To expand this device to an 8-bit adder, you must
   (a) use four adders with no interconnections
   (b) use two adders and connect the sum outputs of one to the bit inputs of the other
   (c) use eight adders with no interconnections
   (d) use two adders with the carry output of one connected to the carry input of the other
8. If a 74HC85 magnitude comparator has $A = 1011$ and $B = 1001$ on its inputs, the outputs are
   (a) $A > B = 0, A < B = 1, A = B = 0$ **(b)** $A > B = 1, A < B = 0, A = B = 0$
   (c) $A > B = 1, A < B = 1, A = B = 0$ **(d)** $A > B = 0, A < B = 0, A = B = 1$
9. If a 1-of-16 decoder with active-LOW outputs exhibits a LOW on the decimal 12 output, what are the inputs?
   (a) $A_3A_2A_1A_0 = 1010$ **(b)** $A_3A_2A_1A_0 = 1110$
   (c) $A_3A_2A_1A_0 = 1100$ **(d)** $A_3A_2A_1A_0 = 0100$
10. A BCD-to-7 segment decoder has 0100 on its inputs. The active outputs are
    (a) *a, c, f, g* **(b)** *b, c, f, g* **(c)** *b, c, e, f* **(d)** *b, d, e, g*
11. If an octal-to-binary priority encoder has its 0, 2, 5, and 6 inputs at the active level, the active-HIGH binary output is
    (a) 110 **(b)** 010 **(c)** 101 **(d)** 000
12. In general, a multiplexer has
    (a) one data input, several data outputs, and selection inputs
    (b) one data input, one data output, and one selection input
    (c) several data inputs, several data outputs, and selection inputs
    (d) several data inputs, one data output, and selection inputs
13. Data selectors are basically the same as
    (a) decoders **(b)** demultiplexers
    (c) multiplexers **(d)** encoders
14. Which of the following codes exhibit even parity?
    (a) 10011000 **(b)** 01111000 **(c)** 11111111
    (d) 11010101 **(e)** all **(f)** both answers (b) and (c)

PROBLEMS

SECTION 6-1 Basic Overview of Logic Functions

1. Name the logic function of each block in Figure 6–70 based on your observation of the inputs and outputs.



▲ FIGURE 6–70

2. A pulse waveform with a frequency of 10 kHz is applied to the input of a counter. During 100 ms, how many pulses are counted?
3. Consider a register that can store eight bits. Assume that it has been reset so that it contains zeros in all positions. If you transfer four alternating bits (0101) serially into the register, beginning with a 1 and shifting to the right, what will the total content of the register be as soon as the fourth bit is stored?

SECTION 6-2 Basic Adders

4. For the full-adder of Figure 6–12, determine the logic state (1 or 0) at each gate output for the following inputs:

   (a) $A = 1, B = 1, C_{in} = 1$ (b) $A = 0, B = 1, C_{in} = 1$ (c) $A = 0, B = 1, C_{in} = 0$

5. What are the full-adder inputs that will produce each of the following outputs:

   (a) $\Sigma = 0, C_{out} = 0$ (b) $\Sigma = 1, C_{out} = 0$

   (c) $\Sigma = 1, C_{out} = 1$ (d) $\Sigma = 0, C_{out} = 1$

6. Determine the outputs of a full-adder for each of the following inputs:

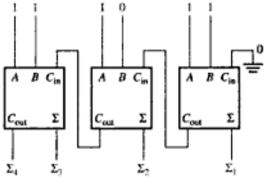
   (a) $A = 1, B = 0, C_{in} = 0$ (b) $A = 0, B = 0, C_{in} = 1$

   (c) $A = 0, B = 1, C_{in} = 1$ (d) $A = 1, B = 1, C_{in} = 1$

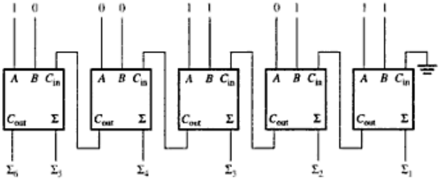
SECTION 6-3 Parallel Binary Adders

7. For the parallel adder in Figure 6–71, determine the complete sum by analysis of the logical operation of the circuit. Verify your result by longhand addition of the two input numbers.
8. Repeat Problem 7 for the circuit and input conditions in Figure 6–72.
9. The input waveforms in Figure 6–73 are applied to a 2-bit adder. Determine the waveforms for the sum and the output carry in relation to the inputs by constructing a timing diagram.

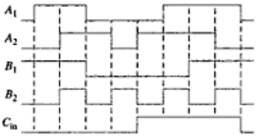
▶ FIGURE 6–71



▶ FIGURE 6–72



▶ FIGURE 6–73



10. The following sequences of bits (right-most bit first) appear on the inputs to a 74LS83A adder. Determine the resulting sequence of bits on each output.

A_1	1001
A_2	1110
A_3	0000
A_4	1011
B_1	1111
B_2	1100
B_3	1010
B_4	0010

11. In the process of checking a 74LS83A 4-bit parallel adder, the following voltage levels are observed on its pins: 1-LOW, 2-HIGH, 3-HIGH, 4-HIGH, 6-HIGH, 7-HIGH, 8-LOW, 9-LOW, 10-LOW, 11-LOW, 13-LOW, 14-HIGH, 15-LOW, and 16-HIGH. Determine if the IC is functioning properly.

SECTION 6–4 Comparators

12. The waveforms in Figure 6–74 are applied to the comparator as shown. Determine the output $(A = B)$ waveform.

13. For the 4-bit comparator in Figure 6–75, plot each output waveform for the inputs shown. The outputs are active-HIGH.

FIGURE 6–74

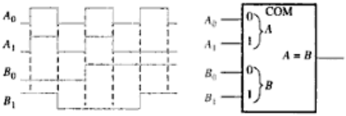
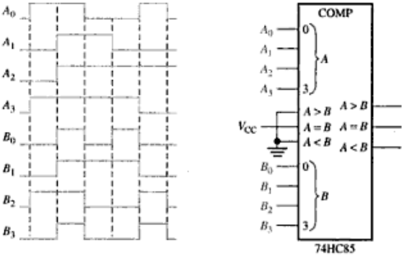

A_0
A_1
B_0
B_1
COM
A_0
A_1
0
1
A
$A = B$
B_0
B_1
0
1
B

FIGURE 6–75


A_0
A_1
A_2
A_3
B_0
B_1
B_2
B_3
COMP
A_0
A_1
A_2
A_3
0
3
A
V_{CC}
$A > B$
$A = B$
$A < B$
$A > B$
$A = B$
$A < B$
B_0
B_1
B_2
B_3
0
3
B
74HC85

14. For each set of binary numbers, determine the output states for the comparator of Figure 6–25.

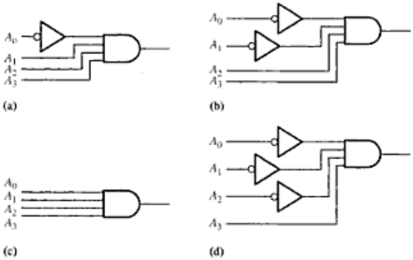
(a) $A_3A_2A_1A_0 = 1100$ **(b)** $A_3A_2A_1A_0 = 1000$ **(c)** $A_3A_2A_1A_0 = 0100$

$B_3B_2B_1B_0 = 1001$ $B_3B_2B_1B_0 = 1011$ $B_3B_2B_1B_0 = 0100$

SECTION 6–5 Decoders

15. When a HIGH is on the output of each of the decoding gates in Figure 6–76, what is the binary code appearing on the inputs? The MSB is A_3.

FIGURE 6–76


A_0
A_1
A_2
A_3
(a)
A_0
A_1
A_2
A_3
(b)
A_0
A_1
A_2
A_3
(c)
A_0
A_1
A_2
A_3
(d)

16. Show the decoding logic for each of the following codes if an active-HIGH (1) output is required:

(a) 1101 **(b)** 1000 **(c)** 11011 **(d)** 11100

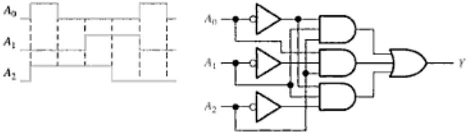
(e) 101010 **(f)** 111110 **(g)** 000101 **(h)** 1110110

17. Solve Problem 16, given that an active-LOW (0) output is required.

18. You wish to detect only the presence of the codes 1010, 1100, 0001, and 1011. An active-HIGH output is required to indicate their presence. Develop the minimum decoding logic with a single output that will indicate when any one of these codes is on the inputs. For any other code, the output must be LOW.

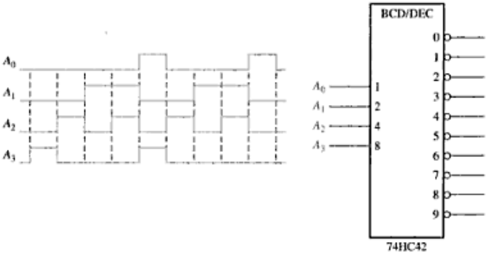
19. If the input waveforms are applied to the decoding logic as indicated in Figure 6–77, sketch the output waveform in proper relation to the inputs.

FIGURE 6–77



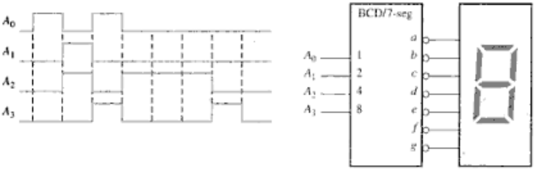
20. BCD numbers are applied sequentially to the BCD-to-decimal decoder in Figure 6–78. Draw a timing diagram, showing each output in the proper relationship with the others and with the inputs.

FIGURE 6–78



21. A 7-segment decoder/driver drives the display in Figure 6–79. If the waveforms are applied as indicated, determine the sequence of digits that appears on the display.

FIGURE 6–79



SECTION 6–6 Encoders

22. For the decimal-to-BCD encoder logic of Figure 6–41, assume that the 9 input and the 3 input are both HIGH. What is the output code? Is it a valid BCD (8421) code?

23. A 74HC147 encoder has LOW levels on pins 2, 5, and 12. What BCD code appears on the outputs if all the other inputs are HIGH?

SECTION 6-7 Code Converters

24. Convert the following decimal numbers to BCD and then to binary.

    (a) 2 (b) 8 (c) 13 (d) 26 (e) 33

25. Show the logic required to convert a 10-bit binary number to Gray code, and use that logic to convert the following binary numbers to Gray code:

    (a) 1010101010 (b) 1111100000 (c) 0000001110 (d) 1111111111

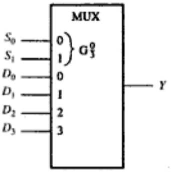
26. Show the logic required to convert a 10-bit Gray code to binary, and use that logic to convert the following Gray code words to binary:

    (a) 1010000000 (b) 0011001100 (c) 1111000111 (d) 0000000001

SECTION 6-8 Multiplexers (Data Selectors)

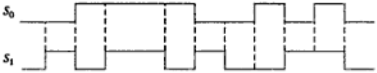
27. For the multiplexer in Figure 6–80, determine the output for the following input states: $D_0 = 0$, $D_1 = 1$, $D_2 = 1$, $D_3 = 0$, $S_0 = 1$, $S_1 = 0$.

► FIGURE 6–80



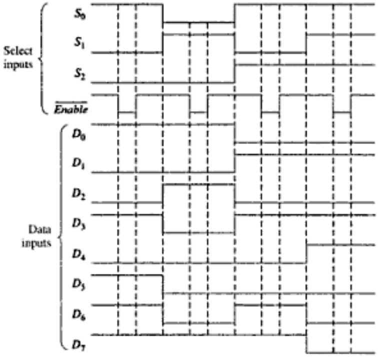
28. If the data-select inputs to the multiplexer in Figure 6–80 are sequenced as shown by the waveforms in Figure 6–81, determine the output waveform with the data inputs specified in Problem 27.

► FIGURE 6–81



29. The waveforms in Figure 6–82 are observed on the inputs of a 74LS151 8-input multiplexer. Sketch the Y output waveform.

► FIGURE 6–82



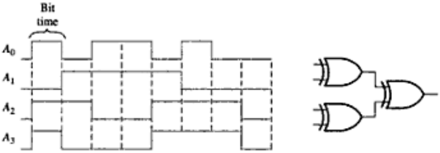
SECTION 6-9 Demultiplexers

30. Develop the total timing diagram (inputs and outputs) for a 74HC154 used in a demultiplexing application in which the inputs are as follows: The data-select inputs are repetitively sequenced through a straight binary count beginning with 0000, and the data input is a serial data stream carrying BCD data representing the decimal number 2468. The least significant digit (8) is first in the sequence, with its LSB first, and it should appear in the first 4-bit positions of the output.

SECTION 6-10 Parity Generators/Checkers

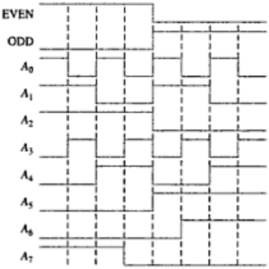
31. The waveforms in Figure 6–83 are applied to the 4-bit parity logic. Determine the output waveform in proper relation to the inputs. For how many bit times does even parity occur, and how is it indicated? The timing diagram includes eight bit times.

▶ **FIGURE 6–83**



32. Determine the Σ Even and the Σ Odd outputs of a 74LS280 9-bit parity generator/checker for the inputs in Figure 6–84. Refer to the truth table in Figure 6–62.

▶ **FIGURE 6–84**



ANSWERS

SECTION REVIEWS

SECTION 6-1 Basic Overview of Logic Functions

1. A comparator compares the magnitudes of two input numbers.
2. Add, subtract, multiply, and divide
3. Encoding is changing a familiar form such as decimal to a coded form such as binary.
4. Decoding is changing a code to a familiar form such as binary to decimal.
5. Multiplexing puts data from many sources onto one line. Demultiplexing takes data from one line and distributes it to many destinations.
6. Flip-flops, registers, semiconductor memories, magnetic disks
7. A counter counts events with a sequence of binary states.

SECTION 6–2 Basic Adders

1. (a) $\Sigma = 1, C_{out} = 0$ (b) $\Sigma = 0, C_{out} = 0$
   (c) $\Sigma = 1, C_{out} = 0$ (d) $\Sigma = 0, C_{out} = 1$
2. $\Sigma = 1, C_{out} = 1$

SECTION 6–3 Parallel Binary Adders

1. $C_{out}\Sigma_4\Sigma_3\Sigma_2\Sigma_1 = 11001$. 2. Three 74LS283s are required to add two 10-bit numbers.

SECTION 6–4 Comparators

1. $A > B = 1, A < B = 0, A = B = 0$ when $A = 1011$ and $B = 1010$
2. Right comparator: pin 7: $A < B = 1$; pin 6: $A = B = 0$; pin 5: $A > B = 0$
   Left comparator: pin 7: $A < B = 0$; pin 6: $A = B = 0$; pin 5: $A > B = 1$

SECTION 6–5 Decoders

1. Output 5 is active when 101 is on the inputs.
2. Four 74HC154s are used to decode a 6-bit binary number.
3. Active-LOW output drives a common-cathode LED display.

SECTION 6–6 Encoders

1. (a) $A_0 = 1, A_1 = 1, A_2 = 0, A_3 = 1$
   (b) No, this is not a valid BCD code.
   (c) Only one input can be active for a valid output.
2. (a) $\overline{A}_3 = 0, \overline{A}_2 = 1, \overline{A}_1 = 1, \overline{A}_0 = 1$
   (b) The output is 0111, which is the complement of 1000 (8).

SECTION 6–7 Code Converters

1. 10000101 (BCD) = 1010101_2
2. An 8-bit binary-to-Gray converter consists of seven exclusive-OR gates in an arrangement like that in Figure 6–46.

SECTION 6–8 Multiplexers (Data Selectors)

1. The output is 0.
2. (a) 74LS157: Quad 2-input data selector
   (b) 74LS151: 8-input data selector
3. The data output alternates between LOW and HIGH as the data-select inputs sequence through the binary states.

4. **(a)** The 74HC157A multiplexes the two BCD codes to the 7-segment decoder.
   (b) The 74LS48 decodes the BCD to energize the display.
   (c) The 74LS139 enables the 7-segment displays alternately.

SECTION 6–9 Demultiplexers

1. A decoder can be used as a multiplexer by using the input lines for data selection and an Enable line for data input.
2. The outputs are all HIGH except D_{10}, which is LOW.

SECTION 6–10 Parity Generators/Checkers

1. **(a)** Even parity: 1110100 **(b)** Even parity: 001100011
2. **(a)** Odd parity: 11010101 **(b)** Odd parity: 11000001
3. **(a)** Code is correct, four 1s.
   (b) Code is in error, seven 1s

SECTION 6–11 Glitches in Decoder Circuits

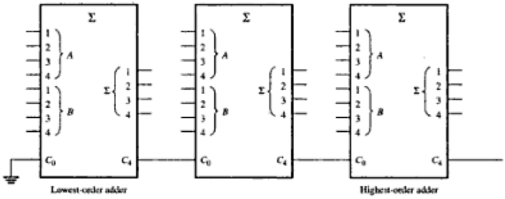
1. A glitch is a very short-duration voltage spike (usually unwanted).
2. Glitches are caused by transition states.
3. Strobe is the enabling of a device for a specified period of time when the device is not in transition.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

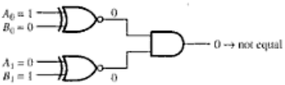
6–1 $\Sigma = 1, C_{out} = 1$ **6–2** $\Sigma_1 = 0, \Sigma_2 = 0, \Sigma_3 = 1, \Sigma_4 = 1$

6–3 1011 + 1010 = 10101 **6–4** See Figure 6–85. **6–5** See Figure 6–86.

▶ **FIGURE 6–85**



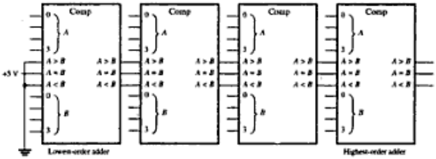
▶ **FIGURE 6–86**

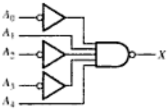


6–6 $A > B = 0, A = B = 0, A < B = 1$ **6–7** See Figure 6–87.

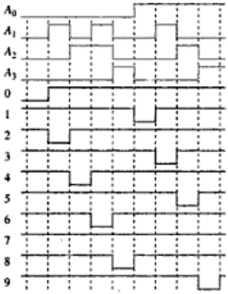
6–8 See Figure 6–88. **6–9** Output 22 **6–10** See Figure 6–89.

▶ **FIGURE 6–87**


Comp
0
A
3
A > B
A = B
A < B
0
B
3
+5 V
Lowest-order adder
Highest-order adder


A_0
A_1
A_2
A_3
A_4
X

▲ **FIGURE 6–88**



▲ **FIGURE 6–89**

6–11 All inputs LOW: $\overline{A}_0 = 0, \overline{A}_1 = 1, \overline{A}_2 = 1, \overline{A}_3 = 0$

All inputs HIGH: All outputs HIGH.

6–12 BCD 01000001

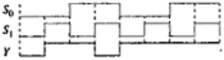
→ 00000001 1

→ 00101000 40

Binary 00101001 41

6–13 Seven exclusive-OR gates **6–14** See Figure 6–90.

▶ **FIGURE 6–90**



6–15 D_0: $S_3 = 0, S_2 = 0, S_1 = 0, S_0 = 0$

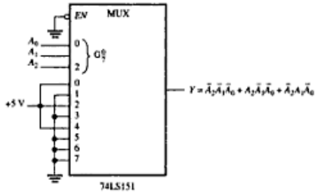
D_4: $S_3 = 0, S_2 = 1, S_1 = 0, S_0 = 0$

D_8: $S_3 = 1, S_2 = 0, S_1 = 0, S_0 = 0$

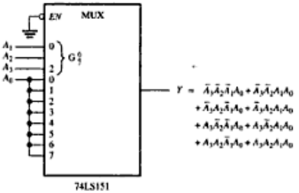
D_{13}: $S_3 = 1, S_2 = 1, S_1 = 0, S_0 = 1$

6–16 See Figure 6–91. **6–17** See Figure 6–92. **6–18** See Figure 6–93.

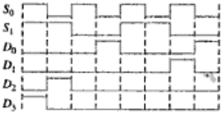
▶ **FIGURE 6–91**


EN
MUX
A_0
A_1
A_2
0
2
G $\frac{0}{7}$
+5 V
0
1
2
3
4
5
6
7
$Y = \bar{A}_2\bar{A}_1\bar{A}_0 + A_2\bar{A}_1\bar{A}_0 + \bar{A}_2A_1\bar{A}_0$
74LS151

▶ **FIGURE 6–92**


EN
MUX
A_1
A_2
A_3
A_0
0
2
G $\frac{0}{7}$
0
1
2
3
4
5
6
7
$Y = \bar{A}_3\bar{A}_2\bar{A}_1A_0 + \bar{A}_3\bar{A}_2A_1A_0$
$+ \bar{A}_3A_2\bar{A}_1A_0 + \bar{A}_3A_2A_1A_0$
$+ A_3\bar{A}_2\bar{A}_1A_0 + A_3\bar{A}_2A_1A_0$
$+ A_3A_2\bar{A}_1A_0 + A_3A_2A_1A_0$
74LS151

▶ **FIGURE 6–93**



SELF-TEST

1. (c) **2.** (e) **3.** (a) **4.** (b) **5.** (c) **6.** (a) **7.** (d) **8.** (b)
9. (c) **10.** (b) **11.** (a) **12.** (d) **13.** (c) **14.** (f)

7

FLIP-FLOPS

CHAPTER OBJECTIVES

- Use logic gates to construct basic latches
- Explain the difference between an S-R latch and a D latch
- Recognize the difference between a latch and a flip-flop
- Explain how S-R, D, and J-K flip-flops differ
- Explain how edge-triggered and master-slave flip-flops differ
- Understand the significance of propagation delays, set-up time, hold time, maximum operating frequency, minimum clock pulse widths, and power dissipation in the application of flip-flops
- Apply flip-flops in basic applications
- Analyze circuits for race conditions and the occurrence of glitches

INTRODUCTION

In the previous chapters, you have studied combinational logic. This chapter begins a study of the fundamentals of sequential logic. Bistable multivibrator devices are covered in this chapter. Two categories of bistable devices are the latch and the flip-flop. Bistable devices have two stable states, called SET and RESET; they can retain either of these states indefinitely, making them useful as storage devices. The basic difference between latches and flip-flops is the way in which they are changed from one state to the other. The flip-flop is a basic building block for counters, registers, and other sequential control logic and is used in certain types of memories. The clock is used as the basic system timing signal for advancing the sequential logic through its states. Sequential logic will be covered in Chapter 8.

7–1 LATCHES

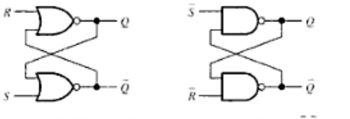
The latch is a type of temporary storage device that has two stable states (bistable) and is normally placed in a category separate from that of flip-flops. Latches are basically similar to flip-flops because they are bistable devices that can reside in either of two states using a feedback arrangement, in which the outputs are connected back to the opposite inputs. The main difference between latches and flip-flops is in the method used for changing their state.

After completing this section, you should be able to

- Explain the operation of a basic S-R latch
- Explain the operation of a gated S-R latch
- Explain the operation of a gated D latch
- Implement an S-R or D latch with logic gates
- Describe the 74LS279 and 74LS75 quad latches

The S-R (SET-RESET) Latch

A **latch** is a type of **bistable** logic device or **multivibrator**. An active-HIGH input S-R (SET-RESET) latch is formed with two cross-coupled NOR gates as shown in Figure 7–1(a); an active-LOW input $\overline{S}$-$\overline{R}$ latch is formed with two cross-coupled NAND gates as shown in Figure 7–1(b). Notice that the output of each gate is connected to an input of the opposite gate. This produces the regenerative **feedback** that is characteristic of all latches and flip-flops.



(a) Active-HIGH input S-R latch (b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

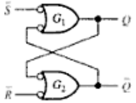
▲ **FIGURE 7–1**

Two versions of SET-RESET (S-R) latches

To explain the operation of the latch, we will use the NAND gate $\overline{S}$-$\overline{R}$ latch in Figure 7–1(b). This latch is redrawn in Figure 7–2 with the negative-OR equivalent symbols used for the NAND gates. This is done because LOWs on the $\overline{S}$ and $\overline{R}$ lines are the activating inputs.

The latch in Figure 7–2 has two inputs, $\overline{S}$ and $\overline{R}$, and two outputs, Q and $\overline{Q}$. Let us start by assuming that both inputs and the Q output are HIGH. Since the Q output is connected back to an input of gate G_2, and the $\overline{R}$ input is HIGH, the output of G_2 must be LOW. This LOW output is coupled back to an input of gate G_1, ensuring that its output is HIGH.

► **FIGURE 7–2**



When the Q output is HIGH, the latch is in the **SET** state. It will remain in this state indefinitely until a LOW is temporarily applied to the $\overline{R}$ input. With a LOW on the $\overline{R}$ input and a HIGH on $\overline{S}$, the output of gate G_2 is forced HIGH. This HIGH on the $\overline{Q}$ output is coupled back to an input of G_1, and since the $\overline{S}$ input is HIGH, the output of G_1 goes LOW. This LOW on the Q output is then coupled back to an input of G_2, ensuring that the $\overline{Q}$ output remains HIGH even when the LOW on the $\overline{R}$ input is removed. When the Q output is LOW, the latch is in the **RESET** state. Now, the latch remains indefinitely in the RESET state until a LOW is applied to the $\overline{S}$ input.

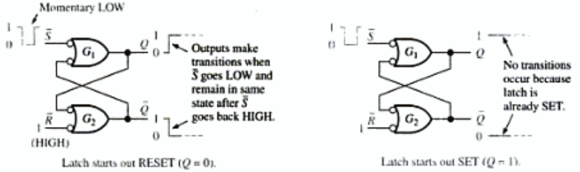
In normal operation, the outputs of a latch are always complements of each other.

When Q is HIGH, $\overline{Q}$ is LOW, and when Q is LOW, $\overline{Q}$ is HIGH

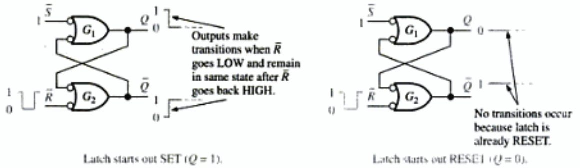
An invalid condition in the operation of an active-LOW input $\overline{S}$-$\overline{R}$ latch occurs when LOWs are applied to both $\overline{S}$ and $\overline{R}$ at the same time. As long as the LOW levels are simultaneously held on the inputs, both the Q and $\overline{Q}$ outputs are forced HIGH, thus violating the basic complementary operation of the outputs. Also, if the LOWs are released simultaneously, both outputs will attempt to go LOW. Since there is always some small difference in the propagation delay time of the gates, one of the gates will dominate in its transition to the LOW output state. This, in turn, forces the output of the slower gate to remain HIGH. In this situation, you cannot reliably predict the next state of the latch.

Figure 7–3 illustrates the active-LOW input $\overline{S}$-$\overline{R}$ latch operation for each of the four possible combinations of levels on the inputs. (The first three combinations are valid, but the last is not.) Table 7–1 summarizes the logic operation in truth table form. Operation of the active-HIGH input NOR gate latch in Figure 7–1(a) is similar but requires the use of opposite logic levels.

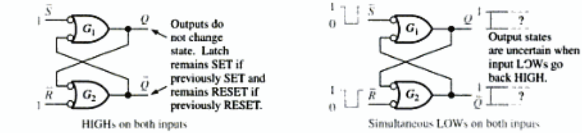
FIGURE 7–3


Momentary LOW
1
0
$\overline{S}$
G_1
Q
1
0
Outputs make transitions when $\overline{S}$ goes LOW and remain in same state after $\overline{S}$ goes back HIGH.
$\overline{R}$
G_2
$\overline{Q}$
1
0
1
(HIGH)
Latch starts out RESET (Q = 0).
1
0
$\overline{S}$
G_1
Q
1
No transitions occur because latch is already SET.
$\overline{R}$
G_2
$\overline{Q}$
0
1
Latch starts out SET (Q = 1).

(a) Two possibilities for the SET operation


$\overline{S}$
1
G_1
Q
1
0
Outputs make transitions when $\overline{R}$ goes LOW and remain in same state after $\overline{R}$ goes back HIGH.
1
0
$\overline{R}$
G_2
$\overline{Q}$
1
0
Latch starts out SET (Q = 1).
$\overline{S}$
1
G_1
Q
0
1
0
$\overline{R}$
G_2
$\overline{Q}$
1
No transitions occur because latch is already RESET.
Latch starts out RESET (Q = 0).

(b) Two possibilities for the RESET operation


$\overline{S}$
1
G_1
Q
Outputs do not change state. Latch remains SET if previously SET and remains RESET if previously RESET.
$\overline{R}$
1
G_2
$\overline{Q}$
HIGHs on both inputs
1
0
$\overline{S}$
G_1
Q
1
?
Output states are uncertain when input LOWs go back HIGH.
1
0
$\overline{R}$
G_2
$\overline{Q}$
1
?
Simultaneous LOWs on both inputs

(c) No-change condition

(d) Invalid condition

TABLE 7–1

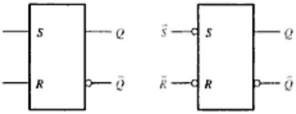
Truth table for $\overline{S}$-$\overline{R}$ latch

INPUTS		OUTPUTS		
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	COMMENTS
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET
1	0	0	1	Latch RESET
0	0	1	1	Invalid condition

Logic symbols for both the active-HIGH input and the active-LOW input latches are shown in Figure 7–4.

FIGURE 7–4

Logic symbols for S-R and $\bar{S}$-$\bar{R}$ latch



(a) Active-HIGH input S-R latch (b) Active-LOW input $\bar{S}$-$\bar{R}$ latch

Example 7–1 illustrates how an active-LOW input $\bar{S}$-$\bar{R}$ latch responds to conditions on its inputs. LOW levels are pulsed on each input in a certain sequence and the resulting Q output waveform is observed. The $\bar{S} = 0$, $\bar{R} = 0$ condition is avoided because it results in an invalid mode of operation and is a major drawback of any SET-RESET type of latch.

EXAMPLE 7–1

If the $\bar{S}$ and $\bar{R}$ waveforms in Figure 7–5(a) are applied to the inputs of the latch in Figure 7–4(b), determine the waveform that will be observed on the Q output. Assume that Q is initially LOW.

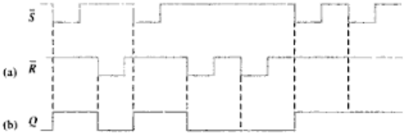


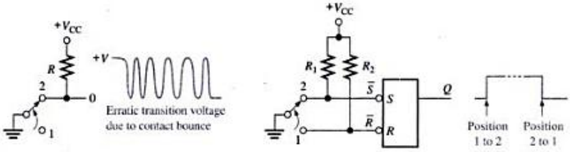
FIGURE 7–5

Solution See Figure 7–5(b).

Supplementary Problem Determine the Q output of an active-HIGH input S-R latch if the waveforms in Figure 7–5(a) are inverted and applied to the inputs.

Application Example

The Latch as a Contact-Bounce Eliminator A good example of an application of an $\bar{S}$-$\bar{R}$ latch is in the elimination of mechanical switch contact "bounce." When the pole of a switch strikes the contact upon switch closure, it physically vibrates or bounces several times before finally making a solid contact. Although these bounces are very short in duration, they produce voltage spikes that are often not acceptable in a digital system. This situation is illustrated in Figure 7–6(a).


+VCC
R
2
0
1
+V
Erratic transition voltage due to contact bounce
+VCC
R1
R2
2
1
S̄
R̄
S
R
Q
Position 1 to 2
Position 2 to 1

(a) Switch contact bounce

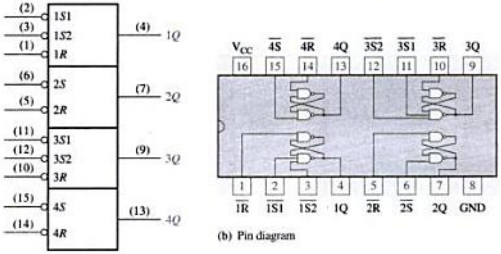
(b) Contact-bounce eliminator circuit

▲ **FIGURE 7–6**

An $\overline{S}$-$\overline{R}$ latch can be used to eliminate the effects of switch bounce as shown in Figure 7–6(b). The switch is normally in position 1, keeping the $\overline{R}$ input LOW and the latch RESET. When the switch is thrown to position 2, $\overline{R}$ goes HIGH because of the pull-up resistor to V_{CC}, and $\overline{S}$ goes LOW on the first contact. Although $\overline{S}$ remains LOW for only a very short time before the switch bounces, this is sufficient to set the latch. Any further voltage spikes on the $\overline{S}$ input due to switch bounce do not affect the latch, and it remains SET. Notice that the Q output of the latch provides a clean transition from LOW to HIGH, thus eliminating the voltage spikes caused by contact bounce. Similarly, a clean transition from HIGH to LOW is made when the switch is thrown back to position 1.

A SET-RESET LATCH

The 74LS279 is a quad $\overline{S}$-$\overline{R}$ latch represented by the logic diagram of Figure 7–7(a) and the pin diagram in part (b). Notice that two of the latches each have two $\overline{S}$ inputs.


(2)
1S1
(3)
1S2
(1)
1R
(4)
1Q
(6)
2S
(5)
2R
(7)
2Q
(11)
3S1
(12)
3S2
(10)
3R
(9)
3Q
(15)
4S
(14)
4R
(13)
4Q
VCC
4S̄
4R̄
4Q
3S̄2
3S̄1
3R̄
3Q
16
15
14
13
12
11
10
9
1
2
3
4
5
6
7
8
1R̄
1S̄1
1S̄2
1Q
2R̄
2S̄
2Q
GND

(a) Logic diagram

(b) Pin diagram

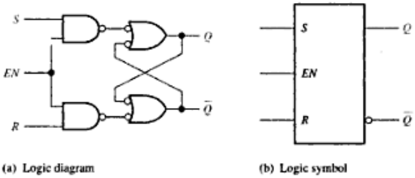
▲ **FIGURE 7–7**

The Gated S-R Latch

A gated latch requires an enable input, *EN* (*G* is also used to designate an enable input). The logic diagram and logic symbol for a gated S-R latch are shown in Figure 7–8. The *S* and *R* inputs control the state to which the latch will go when a HIGH level is applied to the *EN* input. The latch will not change until *EN* is HIGH, but as long as it remains HIGH, the output is controlled by the state of the *S* and *R* inputs. In this circuit, the invalid state occurs when both *S* and *R* are simultaneously HIGH.

▶ FIGURE 7–8

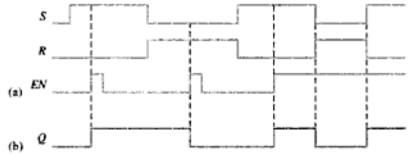
A gated S-R latch



(a) Logic diagram (b) Logic symbol

EXAMPLE 7–2

Determine the *Q* output waveform if the inputs shown in Figure 7–9(a) are applied to a gated S-R latch that is initially RESET.



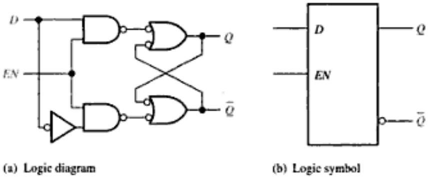
▲ FIGURE 7–9

Solution The *Q* waveform is shown in Figure 7–9(b). Anytime *S* is HIGH and *R* is LOW, a HIGH on the *EN* input sets the latch. Anytime *S* is LOW and *R* is HIGH, a HIGH on the *EN* input resets the latch.

Supplementary Problem Determine the *Q* output of a gated S-R latch if the *S* and *R* inputs in Figure 7–9(a) are inverted.

The Gated D Latch

Another type of gated latch is called the D latch. It differs from the S-R latch because it has only one input in addition to *EN*. This input is called the *D* (data) input. Figure 7–10 contains a logic diagram and logic symbol of a D latch. When the *D* input is HIGH and the *EN* input is HIGH, the latch will set. When the *D* input is LOW, and *EN* is HIGH, the latch will reset. Stated another way, the output *Q* follows the input *D* when *EN* is HIGH.



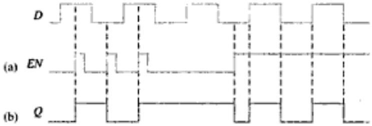
(a) Logic diagram (b) Logic symbol

▲ FIGURE 7–10

A gated D latch

EXAMPLE 7–3

Determine the Q output waveform if the inputs shown in Figure 7–11(a) are applied to a gated D latch, which is initially RESET.



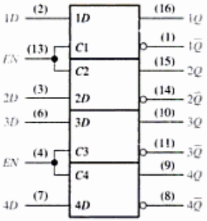
▲ FIGURE 7–11

Solution The Q waveform is shown in Figure 7–11(b). Whenever D is HIGH and EN is HIGH, Q goes HIGH. Whenever D is LOW and EN is HIGH, Q goes LOW. When EN is LOW, the state of the latch is not affected by the D input.

Supplementary Problem Determine the Q output of the gated D latch if the D input in Figure 7–11(a) is inverted.

A D LATCH

An example of a gated D latch is the 74LS75 represented by the logic symbol in Figure 7–12(a). This device has four latches. Notice that each active-HIGH EN input is shared by two latches and is designated as a control input (C). The truth table for each latch is shown in Figure 7–12(b). The X in the truth table represents a "don't care" condition. In this case, when the EN input is LOW, it does not matter what the D input is because the outputs are unaffected and remain in their prior states.



(a) Logic symbol

Inputs		Outputs		
D	EN	Q	$\overline{Q}$	Comments
0	1	0	1	RESET
1	1	1	0	SET
X	0	Q_0	$\overline{Q}_0$	No change

NOTE: Q_0 is the prior output level before the indicated input conditions were established.

(b) Truth table (each latch)

▲ FIGURE 7–12

SECTION 7–1 REVIEW

Answers are at the end of the chapter.

1. List three types of latches.
2. Develop the truth table for the active-HIGH input S-R latch in Figure 7–1(a).
3. What is the Q output of a D latch when $EN = 1$ and $D = 1$?

7–2 EDGE-TRIGGERED FLIP-FLOPS

Flip-flops are synchronous bistable devices, also known as *bistable multivibrators.* In this case, the term *synchronous* means that the output changes state only at a specified point on a triggering input called the *clock* (CLK), which is designated as a control input, *C*; that is, changes in the output occur in synchronization with the clock.

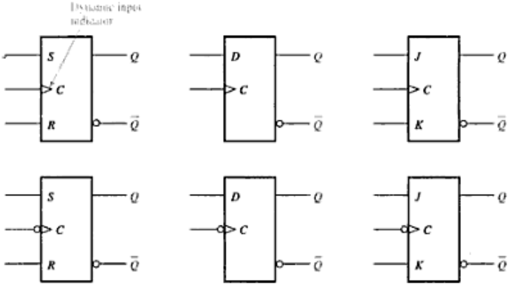
After completing this section, you should be able to

- Define *clock*
- Define *edge-triggered flip-flop*
- Explain the difference between a flip-flop and a latch
- Identify an edge-triggered flip-flop by its logic symbol
- Discuss the difference between a positive and a negative edge-triggered flip-flop
- Discuss and compare the operation of S-R, D, and J-K edge-triggered flip-flops and explain the differences in their truth tables
- Discuss the asynchronous inputs of a flip-flop
- Describe the 74AHC74 and the 74HC112 flip-flops

An **edge-triggered flip-flop** changes state either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse and is sensitive to its inputs only at this transition of the clock. Three types of edge-triggered flip-flops are covered in this section: S-R, D, and J-K. Although the S-R flip-flop is not available in IC form, it is the basis for the D and J-K flip-flops and is, therefore, important to cover. The logic symbols for all of these flip-flops are shown in Figure 7–13. Notice that each type can be either positive edge-triggered (no bubble at *C* input) or negative edge-triggered (bubble at *C* input). The key to identifying an edge-triggered flip-flop by its logic symbol is the small triangle inside the block at the clock (*C*) input. This triangle is called the *dynamic input indicator* ▷.

FIGURE 7–13

Edge-triggered flip-flop logic symbols



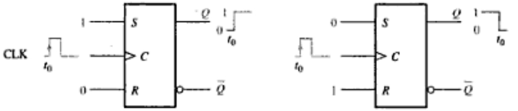
(a) S-R (b) D (c) J-K

The Edge-Triggered S-R Flip-Flop

An S-R flip-flop cannot have both *S* and *R* inputs HIGH at the same time.

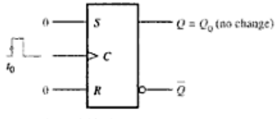
The *S* and *R* inputs of the **S-R flip-flop** are called **synchronous** inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse. When *S* is HIGH and *R* is LOW, the *Q* output goes HIGH on the triggering edge of the clock pulse, and the flip-flop is SET. When *S* is LOW and *R* is HIGH, the *Q* output goes LOW on the triggering edge of the clock pulse, and the flip-flop is RESET. When both *S* and *R* are LOW, the output does not change from its prior state. An invalid condition exists when both *S* and *R* are HIGH.

This basic operation of a positive edge-triggered flip-flop is illustrated in Figure 7–14, and Table 7–2 is the truth table for this type of flip-flop. Remember, *the flip-flop cannot change state except on the triggering edge of a clock pulse.* The *S* and *R* inputs can be changed at any time when the clock input is LOW or HIGH (except for a very short interval around the triggering transition of the clock) without affecting the output.



(a) $S = 1$, $R = 0$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)

(b) $S = 0$, $R = 1$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)



(c) $S = 0$, $R = 0$ flip-flop does not change. (If SET, it remains SET; if RESET, it remains RESET.)

FIGURE 7–14

Operation of a positive edge-triggered S-R flip-flop

TABLE 7–2

Truth table for a positive edge-triggered S-R flip-flop

INPUTS			OUTPUTS		
S	R	CLK	Q	$\overline{Q}$	COMMENTS
0	0	X	Q_0	$\overline{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	?	?	Invalid

↑ = clock transition LOW to HIGH

X = irrelevant ("don't care")

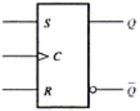
Q_0 = output level prior to clock transition

The operation and truth table for a negative edge-triggered S-R flip-flop are the same as those for a positive edge-triggered device except that the falling edge of the clock pulse is the triggering edge.

EXAMPLE 7–4

Determine the Q and $\overline{Q}$ output waveforms of the flip-flop in Figure 7–15 for the S, R, and CLK inputs in Figure 7–16(a). Assume that the positive edge-triggered flip-flop is initially RESET.

FIGURE 7–15



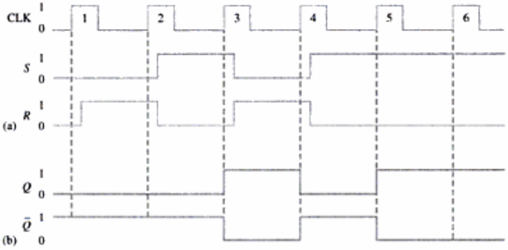


FIGURE 7–16

Solution

1. At clock pulse 1, S is LOW and R is LOW, so Q does not change.
2. At clock pulse 2, S is LOW and R is HIGH, so Q remains LOW (RESET).
3. At clock pulse 3, S is HIGH and R is LOW, so Q goes HIGH (SET).

4. At clock pulse 4, S is LOW and R is HIGH, so Q goes LOW (RESET).
5. At clock pulse 5, S is HIGH and R is LOW, so Q goes HIGH (SET).
6. At clock pulse 6, S is HIGH and R is LOW, so Q stays HIGH.

Once Q is determined, $\overline{Q}$ is easily found since it is simply the complement of Q. The resulting waveforms for Q and $\overline{Q}$ are shown in Figure 7–16(b) for the input waveforms in part (a).

Supplementary Problem Determine Q and $\overline{Q}$ for the S and R inputs in Figure 7–16(a) if the flip-flop is a negative edge-triggered device.

A Method of Edge-Triggering

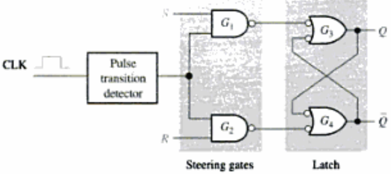
A simplified implementation of an edge-triggered S-R flip-flop is illustrated in Figure 7–17(a) and is used to demonstrate the concept of edge-triggering. This coverage of the S-R flip-flop does not imply that it is the most important type. Actually, the D flip-flop and the J-K flip-flop are available in IC form and more widely used than the S-R type. However, understanding the S-R is important because both the D and the J-K flip-flops are derived from the S-R flip-flop. Notice that the S-R flip-flop differs from the gated S-R latch only in that it has a pulse transition detector. This circuit produces a very short-duration spike on the positive-going transition of the clock pulse.

One basic type of pulse transition detector is shown in Figure 7–17(b). As you can see, there is a small delay on one input to the NAND gate so that the inverted clock pulse arrives at the gate input a few nanoseconds after the true clock pulse. This produces an output spike with a duration of only a few nanoseconds. In a negative edge-triggered flip-flop the clock pulse is inverted first, thus producing a narrow spike on the negative-going edge.

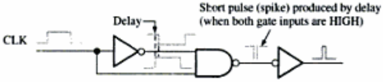
Notice that the circuit in Figure 7–17 is partitioned into two sections, one labeled Steering gates and the other labeled Latch. The steering gates direct, or steer, the clock spike either to the input to gate G_3 or to the input to gate G_4, depending on the state of the S and R inputs. To understand the operation of this flip-flop, begin with the assumptions that it is in the RESET state ($Q = 0$) and that the S, R, and CLK inputs are all LOW. For this condition, the outputs of gate G_1 and gate G_2 are both HIGH. The LOW on the Q output is coupled back into one input of gate G_4, making the $\overline{Q}$ output HIGH. Because $\overline{Q}$ is HIGH, both inputs to gate G_3 are HIGH

FIGURE 7–17

Edge triggering



(a) A simplified logic diagram for a positive edge-triggered S-R flip-flop



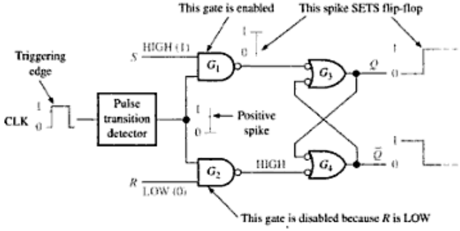
Short pulse (spike) produced by delay (when both gate inputs are HIGH)

(b) A type of pulse transition detector

(remember, the output of gate G_1 is HIGH), holding the Q output LOW. If a pulse is applied to the CLK input, the outputs of gates G_1 and G_2 remain HIGH because they are disabled by the LOWs on the S input and the R input; therefore, there is no change in the state of the flip-flop—it remains in the RESET state.

Let us now make S HIGH, leave R LOW, and apply a clock pulse. Because the S input to gate G_1 is now HIGH, the output of gate G_1 goes LOW for a very short time (spike) when CLK goes HIGH, causing the Q output to go HIGH. Both inputs to gate G_4 are now HIGH (remember, gate G_2 output is HIGH because R is LOW), forcing the $\overline{Q}$ output LOW. This LOW on $\overline{Q}$ is coupled back into one input of gate G_3, ensuring that the Q output will remain HIGH. The flip-flop is now in the SET state. Figure 7–18 illustrates the logic level transitions that take place within the flip-flop for this condition.

▶ **FIGURE 7–18**



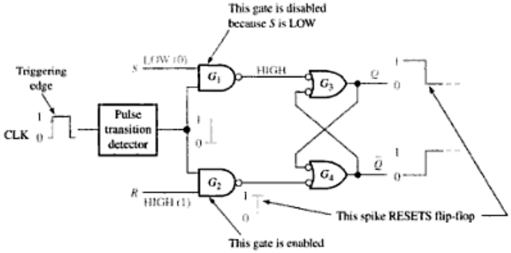
This gate is enabled

This spike SETS flip-flop

This gate is disabled because R is LOW

Next, let us make S LOW and R HIGH and apply a clock pulse. Because the R input is now HIGH, the positive-going edge of the clock produces a negative-going spike on the output of gate G_2, causing the $\overline{Q}$ output to go HIGH. Because of this HIGH on $\overline{Q}$, both inputs to gate G_3 are now HIGH (remember, the output of gate G_1 is HIGH because of the LOW on S), forcing the Q output to go LOW. This LOW on Q is coupled back into one input of gate G_4, ensuring that $\overline{Q}$ will remain HIGH. The flip-flop is now in the RESET state. Figure 7–19 illustrates the logic level transitions that occur within the flip-flop for this condition. As with the gated latch, an invalid condition exists if a clock pulse occurs when both S and R are HIGH at the same time. This is the major drawback of the S-R flip-flop.

▶ **FIGURE 7–19**



This gate is disabled because S is LOW

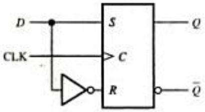
This spike RESETS flip-flop

This gate is enabled

The Edge-Triggered D Flip-Flop

The **D flip-flop** is useful when a single data bit (1 or 0) is to be stored. The addition of an inverter to an S-R flip-flop creates a basic D flip-flop, as in Figure 7–20, where a positive edge-triggered type is shown.

▶ **FIGURE 7–20**



Notice that the flip-flop in Figure 7–20 has only one input, the *D* input, in addition to the clock. If there is a HIGH on the *D* input when a clock pulse is applied, the flip-flop will set, and the HIGH on the *D* input is stored by the flip-flop on the positive-going edge of the clock pulse. If there is a LOW on the *D* input when the clock pulse is applied, the flip-flop will reset, and the LOW on the *D* input is stored by the flip-flop on the leading edge of the clock pulse. In the SET state the flip-flop is storing a 1, and in the RESET state it is storing a 0.

The logical operation of the positive edge-triggered D flip-flop is summarized in Table 7–3. The operation of a negative edge-triggered device is, of course, the same, except that triggering occurs on the falling edge of the clock pulse. *Remember, Q follows D at the active or triggering clock edge.*

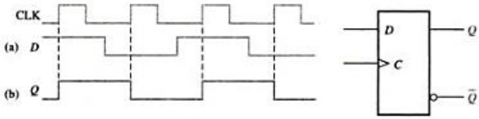
▶ **TABLE 7–3**

INPUTS		OUTPUTS		
D	CLK	*Q*	$\overline{Q}$	COMMENTS
1	↑	1	0	SET (stores a 1)
0	↑	0	1	RESET (stores a 0)

↑ = clock transition LOW to HIGH

EXAMPLE 7–5

Given the waveforms in Figure 7–21(a) for the *D* input and the clock, determine the *Q* output waveform if the flip-flop starts out RESET.



▲ **FIGURE 7–21**

Solution The *Q* output goes to the state of the *D* input at the time of the positive-going clock edge. The resultant output is shown in Figure 7–21(b).

Supplementary Problem Determine the *Q* output for the D flip-flop if the *D* input in Figure 7–21(a) is inverted.

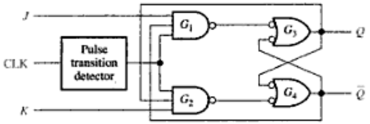
The Edge-Triggered J-K Flip-Flop

The **J-K flip-flop** is versatile and is a widely used type of flip-flop. The J and K designations for the inputs have no known significance except that they are adjacent letters in the alphabet.

The functioning of the J-K flip-flop is identical to that of the S-R flip-flop in the SET, RESET, and no-change conditions of operation. *The difference is that the J-K flip-flop has no invalid state as does the S-R flip-flop.*

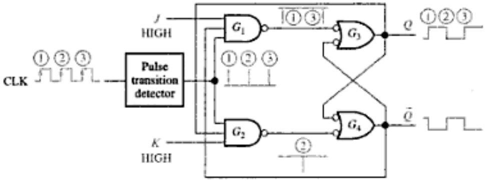
Figure 7–22 shows the basic internal logic for a positive edge-triggered J-K flip-flop. Notice that it differs from the S-R edge-triggered flip-flop in that the Q output is connected back to the input of gate G_2, and the $\overline{Q}$ output is connected back to the input of gate G_1. The two inputs are labeled J and K. A J-K flip-flop can also be of the negative edge-triggered type, in which case the clock input is inverted.

FIGURE 7–22



Let us assume that the flip-flop in Figure 7–23 is RESET and that the J input is HIGH and the K input is LOW rather than as shown. When a clock pulse occurs, a leading-edge spike indicated by ① is passed through gate G_1 because $\overline{Q}$ is HIGH and J is HIGH. This will cause the latch portion of the flip-flop to change to the SET state.

FIGURE 7–23



The flip-flop is now SET. If you now make J LOW and K HIGH, the next clock spike indicated by ② will pass through gate G_2 because Q is HIGH and K is HIGH. This will cause the latch portion of the flip-flop to change to the RESET state.

Now, if a LOW is applied to both the J and K inputs, the flip-flop will stay in its present state when a clock pulse occurs. So, a LOW on both J and K results in a *no-change* condition.

So far, the logical operation of the J-K flip-flop is the same as that of the S-R type in the SET, RESET, and no-change modes. The difference in operation occurs when both the J and K inputs are HIGH. To see this, assume that the flip-flop is RESET. The HIGH on the $\overline{Q}$ enables gate G_1, so the clock spike indicated by ③ passes through to set the flip-flop. Now, there is a HIGH on Q, which allows the next clock spike to pass through gate G_2 and reset the flip-flop.

As you can see, on each successive clock spike, the flip-flop changes to the opposite state. This mode is called **toggle** operation. Figure 7–23 illustrates the transitions when the flip-flop is in the toggle mode. A J-K flip-flop connected for toggle operation is sometimes called a *T flip-flop*.

Table 7–4 summarizes the logical operation of the edge-triggered J-K flip-flop in truth table form. *Notice that there is no invalid state as there is with an S-R flip-flop.* The truth table for a negative edge-triggered device is identical except that it is triggered on the falling edge of the clock pulse.

▶ TABLE 7–4

Truth table for a positive edge-triggered J-K flip-flop

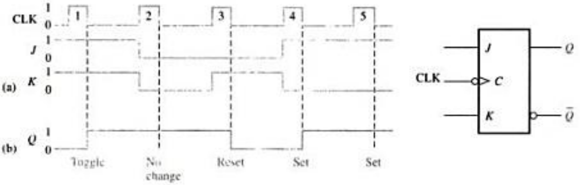
INPUTS			OUTPUTS		
J	K	CLK	Q	$\overline{Q}$	COMMENTS
0	0	↑	Q_0	$\overline{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\overline{Q}_0$	Q_0	Toggle

↑ = clock transition LOW to HIGH

Q_0 = output level prior to clock transition

EXAMPLE 7–6

The waveforms in Figure 7–24(a) are applied to the *J*, *K*, and clock inputs as indicated. Determine the *Q* output, assuming that the flip-flop is initially RESET.



▲ **FIGURE 7–24**

Solution

1. First, since this is a negative edge-triggered flip-flop, as indicated by the "bubble" at the clock input, the *Q* output will change only on the negative-going edge of the clock pulse.
2. At the first clock pulse, both *J* and *K* are HIGH; and because this is a toggle condition, *Q* goes HIGH.
3. At clock pulse 2, a no-change condition exists on the inputs, keeping *Q* at a HIGH level.
4. When clock pulse 3 occurs, *J* is LOW and *K* is HIGH, resulting in a RESET condition; *Q* goes LOW.
5. At clock pulse 4, *J* is HIGH and *K* is LOW, resulting in a SET condition; *Q* goes HIGH.
6. A SET condition still exists on *J* and *K* when clock pulse 5 occurs, so *Q* will remain HIGH.

The resulting *Q* waveform is indicated in Figure 7–24(b).

Supplementary Problem Determine the Q output of the J-K flip-flop if the J and K inputs in Figure 7–24(a) are inverted.

EXAMPLE 7–7

The waveforms in Figure 7–25(a) are applied to the flip-flop as shown. Determine the Q output, starting in the RESET state.

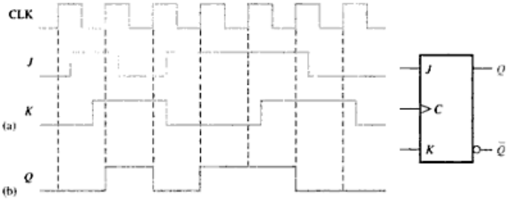


FIGURE 7–25

Solution The Q output assumes the state determined by the states of the J and K inputs at the positive-going edge (triggering edge) of the clock pulse. A change in J or K after the triggering edge of the clock has no effect on the output, as shown in Figure 7–25(b).

Supplementary Problem Interchange the J and K inputs and determine the resulting Q output.

Asynchronous Preset and Clear Inputs

For the flip-flops just discussed, the *S-R*, *D*, and *J-K* inputs are called *synchronous inputs* because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.

Most integrated circuit flip-flops also have **asynchronous** inputs. These are inputs that affect the state of the flip-flop *independent of the clock*. They are normally labeled **preset** (*PRE*) and **clear** (*CLR*), or *direct set* (S_D) and *direct reset* (R_D) by some manufacturers. An active level on the preset input will set the flip-flop, and an active level on the clear input will reset it. A logic symbol for a J-K flip-flop with preset and clear inputs is shown in Figure 7–26. These inputs are active-LOW, as indicated by the bubbles. These preset and clear inputs must both be kept HIGH for synchronous operation.

FIGURE 7–26

Logic symbol for a J-K flip-flop with active-LOW preset and clear inputs

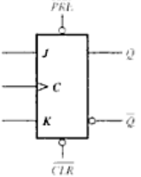
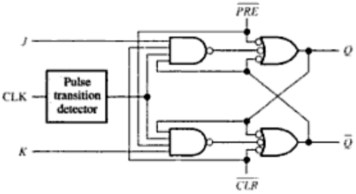


Figure 7–27 shows the logic diagram for an edge-triggered J-K flip-flop with active-LOW preset ($\overline{PRE}$) and clear ($\overline{CLR}$) inputs. This figure illustrates basically how these inputs work. As you can see, they are connected so that they override the effect of the synchronous inputs, *J*, *K*, and the clock.

FIGURE 7–27

Logic diagram for a basic J-K flip-flop with active-LOW preset and clear inputs


PRE
J
Q
CLK
Pulse transition detector
K
Q̄
CLR

EXAMPLE 7–8

For the positive edge-triggered J-K flip-flop with preset and clear inputs in Figure 7–28, determine the *Q* output for the inputs shown in the timing diagram in part (a) if *Q* is initially LOW.

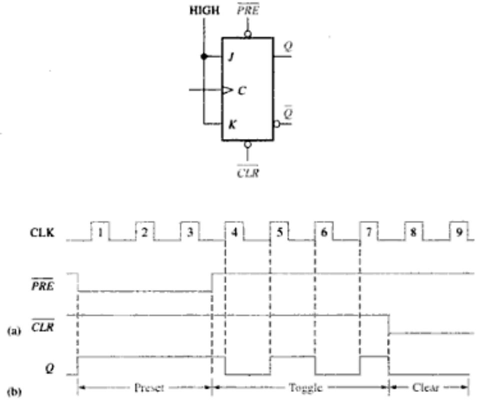
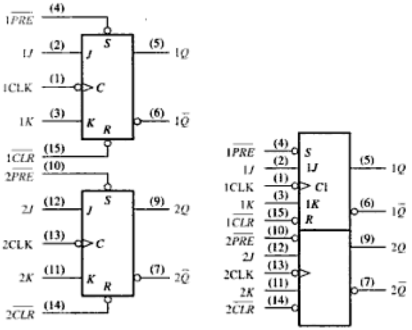

HIGH
PRE
J
C
K
Q
Q̄
CLR
CLK
1
2
3
4
5
6
7
8
9
PRE
(a)
CLR
Q
(b)
Preset
Toggle
Clear

FIGURE 7–28

Solution 1. During clock pulses 1, 2, and 3, the preset ($\overline{PRE}$) is LOW, keeping the flip-flop SET regardless of the synchronous *J* and *K* inputs.

FIGURE 7–30


(4)
1$\overline{PRE}$
1J (2)
1CLK (1)
1K (3)
(5) 1Q
(6) 1$\overline{Q}$
S
J
C
K
R
1$\overline{CLR}$ (15)
2$\overline{PRE}$ (10)
2J (12)
2CLK (13)
2K (11)
2$\overline{CLR}$ (14)
(9) 2Q
(7) 2$\overline{Q}$
1$\overline{PRE}$ (4)
1J (2)
1CLK (1)
1K (3)
1$\overline{CLR}$ (15)
2$\overline{PRE}$ (10)
2J (12)
2CLK (13)
2K (11)
2$\overline{CLR}$ (14)
S
1J
C1
1K
R
(5) 1Q
(6) 1$\overline{Q}$
(9) 2Q
(7) 2$\overline{Q}$

(a) Individual logic symbols (b) Single block logic symbol

EXAMPLE 7–9

The 1J, 1K, 1CLK, 1$\overline{PRE}$, and 1$\overline{CLR}$ waveforms in Figure 7–31(a) are applied to one of the negative edge-triggered flip-flops in a 74HC112 package. Determine the 1Q output waveform.

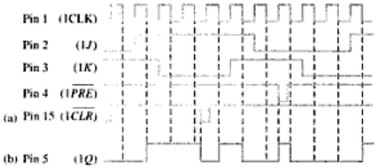

Pin 1 (1CLK)
Pin 2 (1J)
Pin 3 (1K)
Pin 4 (1$\overline{PRE}$)
(a) Pin 15 (1$\overline{CLR}$)
(b) Pin 5 (1Q)

FIGURE 7–31

Solution The resulting 1Q waveform is shown in Figure 7–31(b). Notice that each time a LOW is applied to the 1$\overline{PRE}$ or 1$\overline{CLR}$, the flip-flop is set or reset regardless of the states of the other inputs.

Supplementary Problem Determine the 1Q output waveform if the waveforms for 1$\overline{PRE}$ and 1$\overline{CLR}$ are interchanged.

The master section will assume the state determined by the J and K inputs beginning at the leading (positive-going) edge of the clock pulse. The state of the master section is then transferred to the slave section on the trailing edge of the clock pulse because the outputs of the master are applied to the inputs of the slave and the clock pulse to the slave is inverted. At the trailing edge of the clock pulse, the state of the slave then appears on the Q and $\overline{Q}$ outputs. The Q output is connected back to an input of gate G_2 and the $\overline{Q}$ output is connected back to an input of gate G_1 to produce the characteristic toggle operation when $J = 1$ and $K = 1$. The logic operation is summarized in Table 7–5. One limitation to master-slave operation is that the inputs (J and K) cannot change while the clock pulse is active because the state of the master latch can change during this time.

TABLE 7–5

Truth table for the master-slave J-K flip-flop

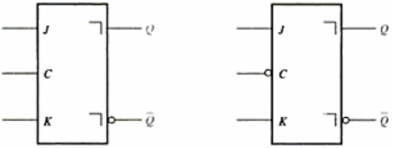
INPUTS			OUTPUTS		
J	K	CLK	Q	$\overline{Q}$	COMMENTS
0	0	⎍	Q_0	$\overline{Q}_0$	No change
0	1	⎍	0	1	RESET
1	0	⎍	1	0	SET
1	1	⎍	$\overline{Q}_0$	Q_0	Toggle

⎍ = clock pulse
Q_0 = output level before clock pulse

The logic symbols for the J-K master-slave flip-flop are shown in Figure 7–33. The key to identifying a pulse-triggered (master-slave) flip-flop by its logic symbol is the ANSI/IEEE *postponed output* symbol (˥) at the outputs. This symbol means that the output does not reflect the J-K input data until the occurrence of the clock edge (either positive-going or negative-going) following the triggering edge. Notice that there is no dynamic input indicator (▷) at the clock (C) input as there is for an edge-triggered flip-flop.

FIGURE 7–33

Pulse-triggered (master-slave) J-K flip-flop logic symbols



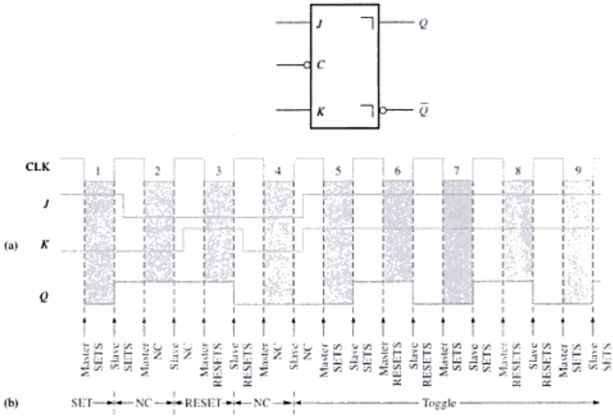
(a) Active-HIGH clock: Data are clocked in on positive-going edge of clock pulse and transferred to output on the following negative-going edge.

(b) Active-LOW clock: Data are clocked in on negative-going edge of clock pulse and transferred to output on the following positive-going edge.

EXAMPLE 7–10

Determine the Q output of the master-slave J-K flip-flop for the input waveforms shown in Figure 7–34(a). The flip-flop starts out RESET and the clock is active-LOW.

Solution The Q waveform is shown in Figure 7–34(b). The input states and events at the beginning and end of each clock pulse are labelled to demonstrate the operation. NC means no change.


J
C
K
Q
$\overline{Q}$
CLK
J
(a)
K
Q
1
2
3
4
5
6
7
8
9
Master SETS
Slave SETS
Master NC
Slave NC
Master RESETS
Slave RESETS
Master NC
Slave NC
Master SETS
Slave SETS
Master RESETS
Slave RESETS
Master SETS
Slave SETS
Master RESETS
Slave RESETS
Master SETS
Slave SETS
(b)
SET
NC
RESET
NC
Toggle

▲ **FIGURE 7–34**

Supplementary Problem What would the Q output look like if the J and K waveforms were inverted?

SECTION 7–3 REVIEW

1. Describe the basic difference between pulse-triggered and edge-triggered flip-flops.
2. Suppose that the D input of a flip-flop changes from LOW to HIGH in the middle of a positive-going clock pulse.
   (a) Describe what happens if the flip-flop is a positive edge-triggered type.
   (b) Describe what happens if the flip-flop is a pulse-triggered master-slave type.

7–4 FLIP-FLOP OPERATING CHARACTERISTICS

The performance, operating requirements, and limitations of flip-flops are specified by several operating characteristics or parameters found on the data sheet for the device. Generally, the specifications are applicable to all CMOS and TTL flip-flops.

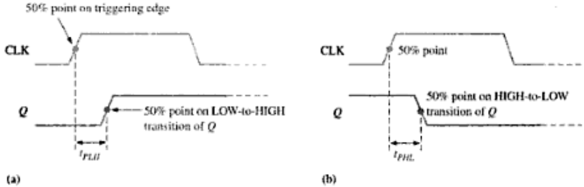
After completing this section, you should be able to

■ Define *propagation delay time* ■ Explain the various propagation delay time specifications ■ Define *set-up time* and discuss how it limits flip-flop operation ■ Define *hold time* and discuss how it limits flip-flop operation ■ Discuss the significance of maximum clock frequency ■ Discuss the various pulse width specifications ■ Define *power dissipation* and calculate its value for a specific device ■ Compare various series of flip-flops in terms of their operating parameters

Propagation Delay Times

A **propagation delay time** is the interval of time required after an input signal has been applied for the resulting output change to occur. Four categories of propagation delay are important in the operation of a flip-flop:

1. Propagation delay t_{PLH} as measured from the triggering edge of the clock pulse to the LOW-to-HIGH transition of the output. This delay is illustrated in Figure 7–35(a).
2. Propagation delay t_{PHL} as measured from the triggering edge of the clock pulse to the HIGH-to-LOW transition of the output. This delay is illustrated in Figure 7–35(b).

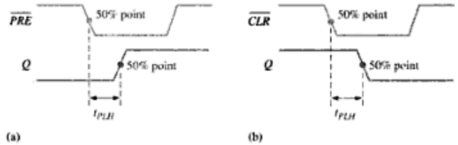

50% point on triggering edge
CLK
Q
50% point on LOW-to-HIGH transition of Q
t_{PLH}
(a)
CLK
50% point
Q
50% point on HIGH-to-LOW transition of Q
t_{PHL}
(b)

▲ **FIGURE 7–35**

Propagation delays, clock to output

3. Propagation delay t_{PLH} as measured from the leading edge of the preset input to the LOW-to-HIGH transition of the output. This delay is illustrated in Figure 7–36(a) for an active-LOW preset input.
4. Propagation delay t_{PHL} as measured from the leading edge of the clear input to the HIGH-to-LOW transition of the output. This delay is illustrated in Figure 7–36(b) for an active-LOW clear input.

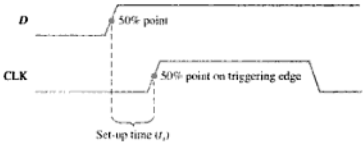
► **FIGURE 7–36**


$\overline{PRE}$
50% point
Q
50% point
t_{PLH}
(a)
$\overline{CLR}$
50% point
Q
50% point
t_{PLH}
(b)

Set-up Time

The **set-up time** (t_s) is the minimum interval required for the logic levels to be maintained constantly on the inputs (*J* and *K*, or *S* and *R*, or *D*) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop. This interval is illustrated in Figure 7–37 for a D flip-flop.

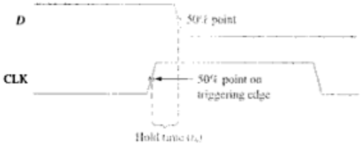
► **FIGURE 7–37**


D
50% point
CLK
50% point on triggering edge
Set-up time (t_s)

Hold time

The **hold time** (t_h) is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop. This is illustrated in Figure 7–38 for a D flip-flop.

FIGURE 7–38



Maximum Clock Frequency

The maximum clock frequency (f_{max}) is the highest rate at which a flip-flop can be reliably triggered. At clock frequencies above the maximum, the flip-flop would be unable to respond quickly enough, and its operation would be impaired.

Pulse Widths

Minimum pulse widths (t_W) for reliable operation are usually specified by the manufacturer for the clock, preset, and clear inputs. Typically, the clock is specified by its minimum HIGH time and its minimum LOW time.

Power Dissipation

The **power dissipation** of any digital circuit is the total power consumption of the device. For example, if the flip-flop operates on a +5 V dc source and draws 5 mA of current, the power dissipation is

$$P = V_{CC} \times I_{CC} = 5\text{ V} \times 5\text{ mA} = 25\text{ mW}$$

The power dissipation is very important in most applications in which the capacity of the dc supply is a concern. As an example, let us assume that you have a digital system that requires a total of ten flip-flops, and each flip-flop dissipates 25 mW of power. The total power requirement is

$$P_T = 10 \times 25\text{ mW} = 250\text{ mW} = 0.25\text{ W}$$

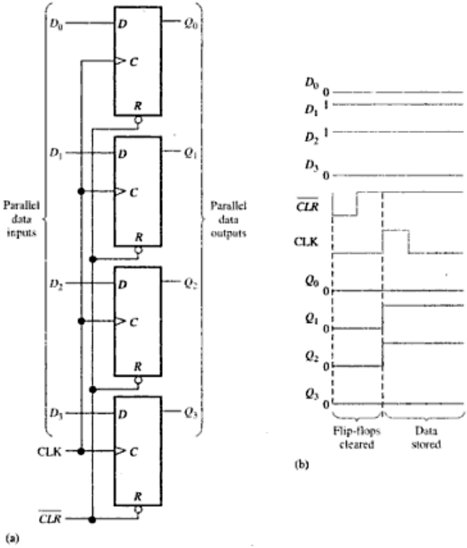
This tells you the output capacity required of the dc supply. If the flip-flops operate on +5 V dc, then the amount of current that the supply must provide is

$$I = \frac{250\text{ mW}}{5\text{ V}} = 50\text{ mA}$$

You must use a +5 V dc supply that is capable of providing at least 50 mA of current.

Comparison of Specific Flip-flops

Table 7–6 provides a comparison, in terms of the operating parameters discussed in this section, of four CMOS and TTL flip-flops of the same type.



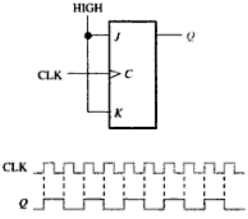
D_0
D_1
D_2
D_3
D
C
R
Q_0
Q_1
Q_2
Q_3
Parallel data inputs
Parallel data outputs
CLK
$\overline{CLR}$
(a)
0
1
1
0
Flip-flops cleared
Data stored
(b)

▲ **FIGURE 7–39**

This group of four flip-flops is an example of a basic register used for data storage. In digital systems, data are normally stored in groups of bits (usually eight or multiples thereof) that represent numbers, codes, or other information. Registers are covered in detail in Chapter 9.

Frequency Division Another application of a flip-flop is dividing (reducing) the frequency of a periodic waveform. When a pulse waveform is applied to the clock input of a J-K flip-flop that is connected to toggle ($J = K = 1$), the Q output is a square wave with one-half the frequency of the clock input. Thus, a single flip-flop can be applied as a divide-by-2 device, as is illustrated in Figure 7–40. As you can see, the flip-flop changes state on each triggering clock edge (positive edge-triggered in this case). This results in an output that changes at half the frequency of the clock waveform.

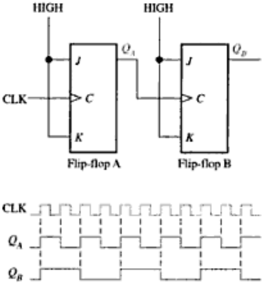
► **FIGURE 7–40**



HIGH
J
C
K
Q
CLK
CLK
Q

Further division of a clock frequency can be achieved by using the output of one flip-flop as the clock input to a second flip-flop, as shown in Figure 7–41. The frequency of the Q_A output is divided by 2 by flip-flop B. The Q_B output is, therefore, one-fourth the frequency of the original clock input. Propagation delay times are not shown on the timing diagrams.

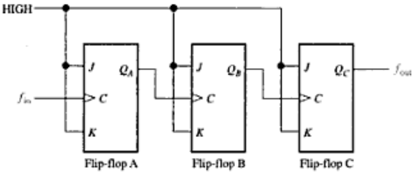
▶ **FIGURE 7–41**



By connecting flip-flops in this way, a frequency division of 2^n is achieved, where n is the number of flip-flops. For example, three flip-flops divide the clock frequency by $2^3 = 8$; four flip-flops divide the clock frequency by $2^4 = 16$; and so on.

EXAMPLE 7–11

Develop the f_{out} waveform for the circuit in Figure 7–42 when an 8 kHz square wave input is applied to the clock input of flip-flop A.

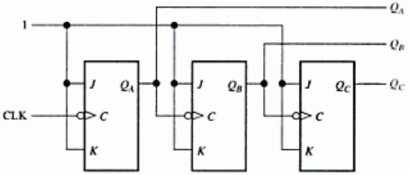


▲ **FIGURE 7–42**

Solution The three flip-flops are connected to divide the input frequency by eight ($2^3 = 8$) and the f_{out} waveform is shown in Figure 7–43. Since these are positive edge-triggered flip-flops, the outputs change on the positive-going clock edge. There is one output pulse for every eight input pulses, so the output frequency is 1 kHz. Waveforms of Q_A and Q_B are also shown.

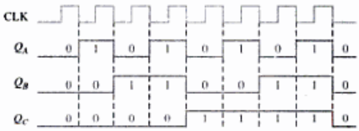
EXAMPLE 7–12

Determine the output waveforms in relation to the clock for Q_A, Q_B, and Q_C in the circuit of Figure 7–45 and show the binary sequence represented by these waveforms.



▲ FIGURE 7–45

Solution The output timing diagram is shown in Figure 7–46. Notice that the outputs change on the negative-going edge of the clock pulses. The outputs go through the binary sequence 000, 001, 010, 011, 100, 101, 110, and 111 as indicated.



▲ FIGURE 7–46

Supplementary Problem How many flip-flops are required to produce a binary sequence representing decimal numbers 0 through 15?

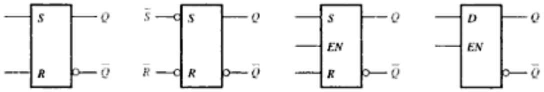
SECTION 7–5 REVIEW

1. A group of flip-flops used for data storage is called a ________.
2. How must a J-K flip-flop be connected to function as a divide-by-2 device?
3. How many flip-flops are required to produce a divide-by-64 device?

SUMMARY

- Symbols for latches and flip-flops are shown in Figure 7–47.

FIGURE 7–47

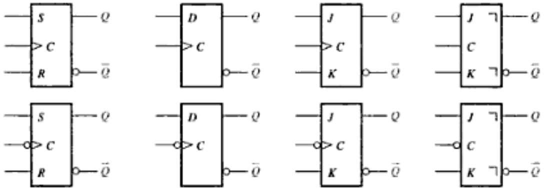


(a) Active-HIGH input S-R latch

(b) Active-LOW input $\overline{S}$-$\overline{R}$ latch

(c) Gated S-R latch

(d) Gated D latch



(e) S-R edge-triggered flip-flops

(f) D edge-triggered flip-flops

(g) J-K edge-triggered flip-flops

(h) J-K master-slave flip-flops

- Latches are bistable devices whose state normally depends on asynchronous inputs.
- Edge-triggered flip-flops are bistable devices with synchronous inputs whose state depends on the inputs only at the triggering transition of a clock pulse. Changes in the outputs occur at the triggering transition of the clock.
- Pulse-triggered master-slave flip-flops are bistable devices with synchronous inputs whose state depends on the inputs at the leading edge of the clock pulse, but whose output is postponed and does not reflect the internal state until the trailing edge of the clock pulse. The synchronous inputs should not be allowed to change while the clock is HIGH.

SELF-TEST

Answers are at the end of the chapter.

1. If an S-R latch has a 1 on the *S* input and a 0 on the *R* input and then the *S* input goes to 0, the latch will be
   (a) set **(b)** reset **(c)** invalid **(d)** clear
2. The invalid state of an S-R latch occurs when
   (a) $S = 1, R = 0$ **(b)** $S = 0, R = 1$
   (c) $S = 1, R = 1$ **(d)** $S = 0, R = 0$
3. For a gated D latch, the *Q* output always equals the *D* input
   (a) before the enable pulse **(b)** during the enable pulse
   (c) immediately after the enable pulse **(d)** answers (b) and (c)
4. Like the latch, the flip-flop belongs to a category of logic circuits known as
   (a) monostable multivibrators **(b)** bistable multivibrators
   (c) astable multivibrators **(d)** one-shots
5. The purpose of the clock input to a flip-flop is to
   (a) clear the device
   (b) set the device
   (c) always cause the output to change states
   (d) cause the output to assume a state dependent on the controlling (*S-R*, *J-K*, or *D*) inputs.

6. For an edge-triggered D flip-flop,
   - **(a)** a change in the state of the flip-flop can occur only at a clock pulse edge
   - **(b)** the state that the flip-flop goes to depends on the D input
   - **(c)** the output follows the input at each clock pulse
   - **(d)** all of these answers
7. A feature that distinguishes the J-K flip-flop from the S-R flip-flop is the
   - **(a)** toggle condition
   - **(b)** preset input
   - **(c)** type of clock
   - **(d)** clear input
8. A flip-flop is in the toggle condition when
   - **(a)** $J = 1, K = 0$
   - **(b)** $J = 1, K = 1$
   - **(c)** $J = 0, K = 0$
   - **(d)** $J = 0, K = 1$
9. A J-K flip-flop with $J = 1$ and $K = 1$ has a 10 kHz clock input. The Q output is
   - **(a)** constantly HIGH
   - **(b)** constantly LOW
   - **(c)** a 10 kHz square wave
   - **(d)** a 5 kHz square wave

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 7–1 Latches

1. If the waveforms in Figure 7–48 are applied to an active-LOW input S-R latch, draw the resulting Q output waveform in relation to the inputs. Assume that Q starts LOW.
2. Solve Problem 1 for the input waveforms in Figure 7–49 applied to an active-HIGH S-R latch.
3. Solve Problem 1 for the input waveforms in Figure 7–50.

FIGURE 7–48

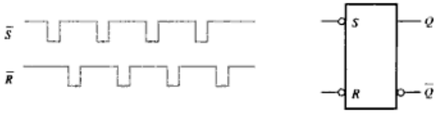


FIGURE 7–49

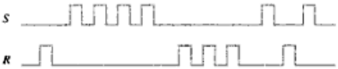
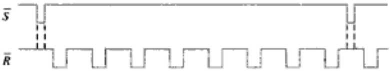
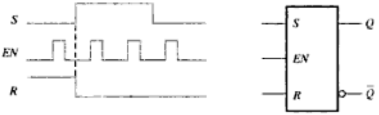


FIGURE 7–50



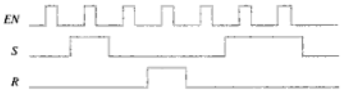
4. For a gated S-R latch, determine the Q and $\overline{Q}$ outputs for the inputs in Figure 7–51. Show them in proper relation to the enable input. Assume that Q starts LOW.

FIGURE 7–51



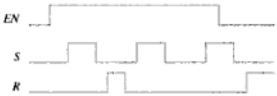
5. Solve Problem 4 for the inputs in Figure 7–52.

▶ **FIGURE 7–52**



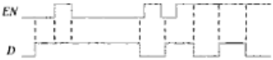
6. Solve Problem 4 for the inputs in Figure 7–53.

▶ **FIGURE 7–53**



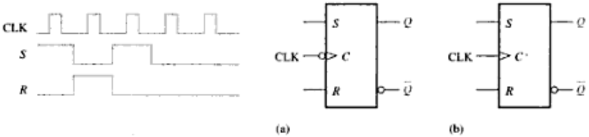
7. For a gated D latch, the waveforms shown in Figure 7–54 are observed on its inputs. Draw the timing diagram showing the output waveform you would expect to see at Q if the latch is initially RESET.

▶ **FIGURE 7–54**



SECTION 7–2 Edge-Triggered Flip-Flops

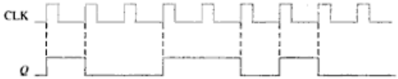
8. Two edge-triggered S-R flip-flops are shown in Figure 7–55. If the inputs are as shown, draw the Q output of each flip-flop relative to the clock, and explain the difference between the two. The flip-flops are initially RESET.



▲ **FIGURE 7–55**

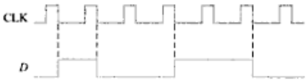
9. The Q output of an edge-triggered S-R flip-flop is shown in relation to the clock signal in Figure 7–56. Determine the input waveforms on the S and R inputs that are required to produce this output if the flip-flop is a positive edge-triggered type.

▶ **FIGURE 7–56**



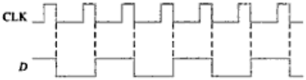
10. Draw the Q output relative to the clock for a D flip-flop with the inputs as shown in Figure 7–57. Assume positive edge-triggering and Q initially LOW.

▶ **FIGURE 7–57**



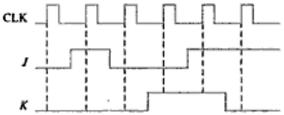
11. Solve Problem 10 for the inputs in Figure 7–58.

▸ FIGURE 7–58



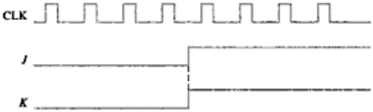
12. For a positive edge-triggered J-K flip-flop with inputs as shown in Figure 7–59, determine the Q output relative to the clock. Assume that Q starts LOW.

▸ FIGURE 7–59



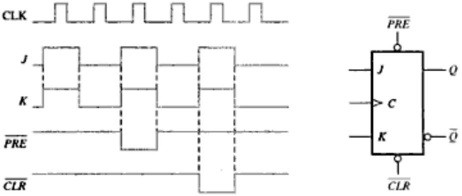
13. Solve Problem 12 for the inputs in Figure 7–60.

▸ FIGURE 7–60



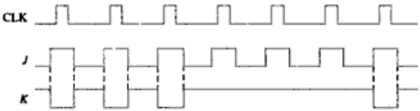
14. Determine the Q waveform relative to the clock if the signals shown in Figure 7–61 are applied to the inputs of the J-K flip-flop. Assume that Q is initially LOW.

▸ FIGURE 7–61



15. For a negative edge-triggered J-K flip-flop with the inputs in Figure 7–62, develop the Q output waveform relative to the clock. Assume that Q is initially LOW.

▸ FIGURE 7–62



16. The following serial data are applied to the flip-flop through the AND gates as indicated in Figure 7–63. Determine the resulting serial data that appear on the Q output. There is one clock pulse for each bit time. Assume that Q is initially 0 and that $\overline{PRE}$ and $\overline{CLR}$ are HIGH. Rightmost bits are applied first.

J_1: 1 0 1 0 0 1 1

J_2: 0 1 1 1 0 1 0

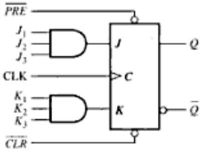
J_3: 1 1 1 1 0 0 0

K_1: 0 0 0 1 1 1 0

K_2: 1 1 0 1 1 0 0

K_3: 1 0 1 0 1 0 1

FIGURE 7–63


$\overline{PRE}$
J_1
J_2
J_3
J
Q
CLK
C
K_1
K_2
K_3
K
$\overline{Q}$
$\overline{CLR}$

17. For the circuit in Figure 7–63, complete the timing diagram in Figure 7–64 by showing the Q output (which is initially LOW). Assume $\overline{PRE}$ and $\overline{CLR}$ remain HIGH.

18. Solve Problem 17 with the same J and K inputs but with the $\overline{PRE}$ and $\overline{CLR}$ inputs as shown in Figure 7–65 in relation to the clock.

FIGURE 7–64

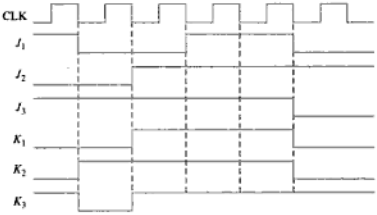
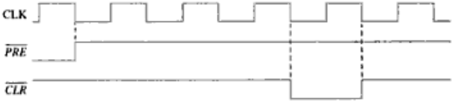

CLK
J_1
J_2
J_3
K_1
K_2
K_3

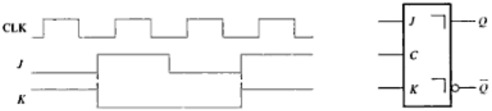
FIGURE 7–65


CLK
$\overline{PRE}$
$\overline{CLR}$

SECTION 7–3 Master-Slave Flip-Flops

19. Determine the Q output in Figure 7–66. Assume that Q is initially LOW.

FIGURE 7–66


CLK
J
K
J
C
K
Q
$\overline{Q}$

SECTION 7–3 Master-Slave Flip-Flops

1. In the pulse-triggered flip-flop (master-slave), a data bit goes into the master section on the leading edge of a clock pulse and is transferred to the output (slave) on the trailing edge. In the edge-triggered flip-flop, a data bit goes into the flip-flop and appears on the output on the same clock edge.
2. **(a)** Nothing happens for the positive edge-triggered flip-flop.
   (b) The output of a pulse-triggered master-slave flip-flop will change.

SECTION 7–4 Flip-Flop Operating Characteristics

1. **(a)** Set-up time is the time required for input data to be present before the triggering edge of the clock pulse.
   (b) Hold time is the time required for data to remain on input after the triggering edge of the clock pulse.
2. The 74AHC74 can be operated at the highest frequency, according to Table 7–6.

SECTION 7–5 Flip-Flop Applications

1. A group of data storage flip-flops is a register.
2. For divide-by-2 operation, the flip-flop must toggle ($J = 1, K = 1$).
3. Six flip-flops are used in a divide-by-64 device.

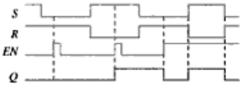
SUPPLEMENTARY PROBLEMS FOR EXAMPLES

7–1 The Q output is the same as shown in Figure 7–5(b).

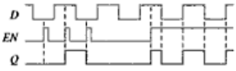
7–2 See Figure 7–73. **7–3** See Figure 7–74.

7–4 See Figure 7–75. **7–5** See Figure 7–76.

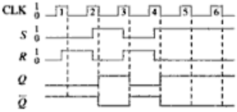
7–6 See Figure 7–77. **7–7** See Figure 7–78.



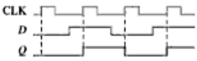
▲ FIGURE 7–73



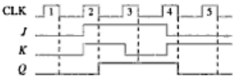
▲ FIGURE 7–74



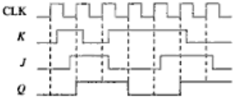
▲ FIGURE 7–75



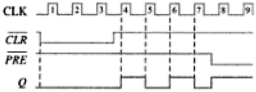
▲ FIGURE 7–76



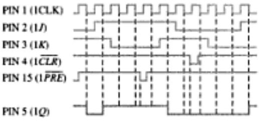
▲ FIGURE 7–77



▲ FIGURE 7–78


CLK
1
2
3
4
5
6
7
8
9
$\overline{CLR}$
$\overline{PRE}$
Q

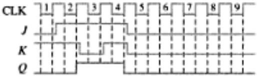
▲ **FIGURE 7–79**


PIN 1 (1CLK)
PIN 2 (1J)
PIN 3 (1K)
PIN 4 (1$\overline{CLR}$)
PIN 15 (1$\overline{PRE}$)
PIN 5 (1Q)

▲ **FIGURE 7–80**

7–8 See Figure 7–79. **7–9** See Figure 7–80.

► **FIGURE 7–81**


CLK
1
2
3
4
5
6
7
8
9
J
K
Q

7–10 See Figure 7–81.

7–11 $2^5 = 32$. Five flip-flops are required.

7–12 Sixteen states require four flip-flops ($2^4 = 16$).

SELF-TEST

1. (a) **2.** (c) **3.** (d) **4.** (b) **5.** (d) **6.** (d) **7.** (a) **8.** (b)
9. (d)

8

COUNTERS

CHAPTER OBJECTIVES

- Describe the difference between an asynchronous and a synchronous counter
- Analyze counter timing diagrams
- Analyze counter circuits
- Explain how propagation delays affect the operation of a counter
- Determine the modulus of a counter
- Modify the modulus of a counter
- Recognize the difference between a 4-bit binary counter and a decade counter
- Use an up/down counter to generate forward and reverse binary sequences
- Determine the sequence of a counter
- Use IC counters in various applications
- Design a counter that will have any specified sequence of states
- Use cascaded counters to achieve a higher modulus
- Use logic gates to decode any given state of a counter
- Eliminate glitches in counter decoding
- Explain how a digital clock operates

INTRODUCTION

As you learned in Chapter 7, flip-flops can be connected together to perform counting operations. Such a group of flip-flops is a counter. The number of flip-flops used and the way in which they are connected determine the number of states (called the modulus) and also the specific sequence of states that the counter goes through during each complete cycle.

Counters are classified into two broad categories according to the way they are clocked: asynchronous and synchronous. In asynchronous counters, commonly called *ripple counters*, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. In synchronous counters, the clock input is connected to all of the flip-flops so that they are clocked simultaneously. Within each of these two categories, counters are classified primarily by the type of sequence, the number of states, or the number of flip-flops in the counter.

8–1 ASYNCHRONOUS COUNTER OPERATION

The term *asynchronous* refers to events that do not have a fixed time relationship with each other and, generally, do not occur at the same time. An asynchronous counter is one in which the flip-flops (FF) within the counter do not change states at exactly the same time because they do not have a common clock pulse.

After completing this section, you should be able to

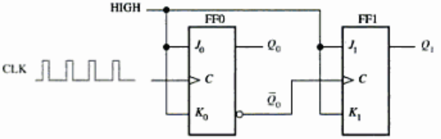
- Describe the operation of a 2-bit asynchronous binary counter
- Describe the operation of a 3-bit asynchronous binary counter
- Define *ripple* in relation to counters
- Describe the operation of an asynchronous decade counter
- Develop counter timing diagrams
- Discuss the 74LS93A 4-bit asynchronous binary counter

A 2-Bit Asynchronous Binary Counter

Figure 8–1 shows a 2-bit counter connected for **asynchronous** operation. Notice that the clock (CLK) is applied to the clock input (*C*) of *only* the first flop-flop, FF0, which is always the least significant bit (LSB). The second flip-flop, FF1, is triggered by the $\overline{Q}_0$ output of FF0. FF0 changes state at the positive-going edge of each clock pulse, but FF1 changes only when triggered by a positive-going transition of the $\overline{Q}_0$ output of FF0. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse (CLK) and a transition of the $\overline{Q}_0$ output of FF0 can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, so the counter operation is asynchronous. The asynchronous counters are also known as ripple counters.

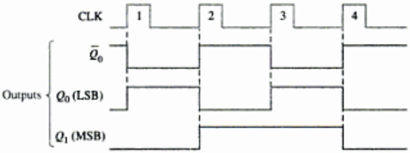
FIGURE 8–1

A 2-bit asynchronous binary counter



The Timing Diagram Let us examine the basic operation of the **asynchronous counter** of Figure 8–1 by applying four clock pulses to FF0 and observing the *Q* output of each flip-flop. Figure 8–2 illustrates the changes in the state of the flip-flop outputs in response to the clock pulses. Both flip-flops are connected for toggle operation ($J = 1$, $K = 1$) and are assumed to be initially RESET (*Q* LOW).

FIGURE 8–2



The positive-going edge of CLK1 (clock pulse 1) causes the Q_0 output of FF0 to go HIGH, as shown in Figure 8–2. At the same time the $\overline{Q}_0$ output goes LOW, but it has no effect on FF1 because a positive-going transition must occur to trigger the flip-flop. After the leading edge of CLK1, $Q_0 = 1$ and $Q_1 = 0$. The positive-going edge of CLK2 causes Q_0 to go LOW. Output $\overline{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go HIGH. After the leading edge of CLK2, $Q_0 = 0$ and $Q_1 = 1$. The positive-going edge of CLK3 causes Q_0 to go HIGH again.

Output $\overline{Q}_0$ goes LOW and has no effect on FF1. Thus, after the leading edge of CLK3, $Q_0 = 1$ and $Q_1 = 1$. The positive-going edge of CLK4 causes Q_0 to go LOW, while $\overline{Q}_0$ goes HIGH and triggers FF1, causing Q_1 to go LOW. After the leading edge of CLK4, $Q_0 = 0$ and $Q_1 = 0$. The counter has now recycled to its original state (both flip-flops are RESET).

In the timing diagram, the waveforms of the Q_0 and Q_1 outputs are shown relative to the clock pulses as illustrated in Figure 8–2. For simplicity, the transitions of Q_0, Q_1, and the clock pulses are shown as simultaneous even though this is an asynchronous counter. There is, of course, some small delay between the CLK and the Q_0 transition and between the $\overline{Q}_0$ transition and the Q_1 transition.

Note in Figure 8–2 that the 2-bit counter exhibits four different states, as you would expect with two flip-flops ($2^2 = 4$). Also, notice that if Q_0 represents the least significant bit (LSB) and Q_1 represents the most significant bit (MSB), the sequence of counter states represents a sequence of binary numbers as listed in Table 8–1. *In digital logic, Q_0 is always the LSB unless otherwise specified.*

TABLE 8–1

Binary state sequence for the counter in Figure 8–1

CLOCK PULSE	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Since it goes through a **binary sequence**, the counter in Figure 8–1 is a binary counter. It actually counts the number of clock pulses up to three, and on the fourth pulse it recycles to its original state ($Q_0 = 0$, $Q_1 = 0$). The term **recycle** is commonly applied to counter operation; it refers to the transition of the counter from its *final state* back to its *original state*.

A 3-Bit Asynchronous Binary Counter

A 3-bit asynchronous binary counter is shown in Figure 8–3(a). The basic operation is the same as that of the 2-bit counter just discussed, except that the 3-bit counter has eight states, due to its three flip-flops. A timing diagram is shown in Figure 8–3(b) for eight clock pulses. Notice that the counter in Figure 8–3 progresses through a binary count of zero through seven and then recycles to the zero state. This counter sequence is listed in Table 8–2. This counter can be easily expanded for higher count, by connecting additional toggle flip-flops.

TABLE 8–2

CLOCK PULSE	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

Asynchronous Decade Counters

The **modulus** of a counter is the number of unique states that the counter will sequence through. The maximum possible number of states (maximum modulus) of a counter is 2^n, where n is the number of flip-flops in the counter.

Counters can also be designed to have a number of states in their sequence that is less than the maximum of 2^n. The resulting sequence is called a **truncated sequence.**

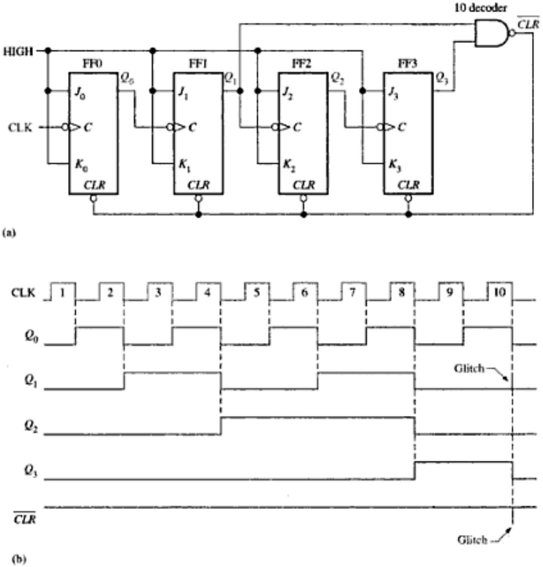
One common modulus for counters with **truncated** sequences is ten (called MOD10). Counters with ten states in their sequence are called **decade counters**. A **decade** counter with a count sequence of zero (0000) through nine (1001) is a BCD decade counter because its ten-state sequence produces the BCD code. This type of counter is useful in display applications in which BCD is required for conversion to a decimal readout.

To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. For example, the BCD decade counter must recycle back to the 0000 state after the 1001 state. A decade counter requires four flip-flops (three flip-flops are insufficient because $2^3 = 8$).

Let us use a 4-bit asynchronous counter such as the one in Figure 8–5(a) and modify its sequence to illustrate the principle of truncated counters. One way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to the clear ($\overline{CLR}$) inputs of the flip-flops, as shown in Figure 8–6(a).

Partial Decoding Notice in Figure 8–6(a) that only Q_1 and Q_3 are connected to the NAND gate inputs. This arrangement is an example of *partial decoding*, in which the two unique states ($Q_1 = 1$ and $Q_3 = 1$) are sufficient to decode the count of ten because none of the other states (zero through nine) have both Q_1 and Q_3 HIGH at the same time. When the counter goes

▸ FIGURE 8–6



into count ten (1010), the decoding gate output goes LOW and asynchronously resets all the flip-flops.

The resulting timing diagram is shown in Figure 8–6(b). Notice that there is a glitch on the Q_1 waveform. The reason for this glitch is that Q_1 must first go HIGH before the count of ten can be decoded. Not until several nanoseconds after the counter goes to the count of ten does the output of the decoding gate go LOW (both inputs are HIGH). Thus, the counter is in the 1010 state for a short time before it is reset to 0000, thus producing the glitch on Q_1 and the resulting glitch on the $\overline{CLR}$ line that resets the counter.

Other truncated sequences can be implemented in a similar way, as Example 8–2 shows.

EXAMPLE 8–2

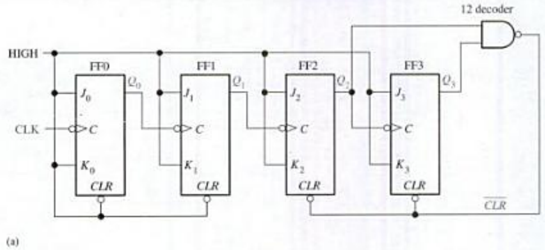
Show how an asynchronous counter can be implemented having a modulus of twelve with a straight binary sequence from 0000 through 1011.

Solution Since three flip-flops can produce a maximum of eight states, four flip-flops are required to produce any modulus greater than eight but less than or equal to sixteen.

When the counter gets to its last state, 1011, it must recycle back to 0000 rather than going to its normal next state of 1100, as illustrated in the following sequence chart:

Q_3	Q_2	Q_1	Q_0	
0	0	0	0	
.	.	.	.	
.	.	.	.	Recycles
.	.	.	.	
1	0	1	1	
1	1	0	0	Normal next state

Observe that Q_0 and Q_1 both go to 0 anyway, but Q_2 and Q_3 must be forced to 0 on the twelfth clock pulse. Figure 8–7(a) shows the modulus-12 counter. The NAND gate partially decodes count twelve (1100) and resets flip-flop 2 and flip-flop 3. Thus, on the twelfth clock pulse, the counter is forced to recycle from count eleven to count zero, as shown in the timing diagram of Figure 8–7(b). (It is in count twelve for only a few nanoseconds before it is reset by the glitch on $\overline{CLR}$.)



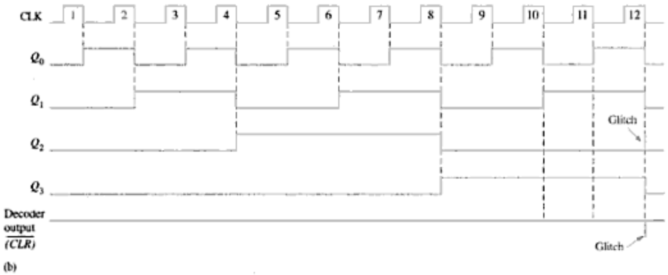


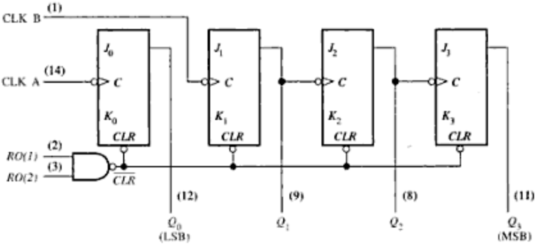
FIGURE 8–7

Supplementary Problem How can the counter in Figure 8–7(a) be modified to make it a modulus-13 counter?

A 4-BIT ASYNCHRONOUS BINARY COUNTER

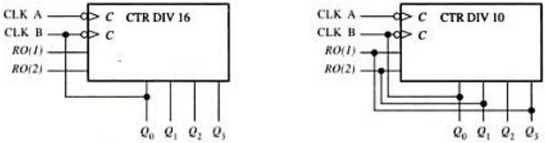
The 74LS93A is an example of a specific integrated circuit asynchronous counter. As the logic diagram in Figure 8–8 shows, this device actually consists of a single flip-flop and a 3-bit asynchronous counter. This arrangement is for flexibility. It can be used as a divide-by-2 device if only the single flip-flop is used, or it can be used as a modulus-8 counter if only the 3-bit counter portion is used. This device also provides gated reset inputs, *RO(1)* and *RO(2)*. When both of these inputs are HIGH, the counter is reset to the 0000 state $\overline{CLR}$.

FIGURE 8–8


CLK B
(1)
CLK A
(14)
J_0
C
K_0
CLR
J_1
C
K_1
CLR
J_2
C
K_2
CLR
J_3
C
K_3
CLR
(2)
$RO(1)$
(3)
$RO(2)$
$\overline{CLR}$
(12)
(9)
(8)
(11)
Q_0 (LSB)
Q_1
Q_2
Q_3 (MSB)

Additionally, the 74LS93A can be used as a 4-bit modulus-16 counter (counts 0 through 15) by connecting the Q_0 output to the CLK B input as shown in Figure 8–9(a). It can also be configured as a decade counter (counts 0 through 9) with asynchronous recycling by using the gated reset inputs for partial decoding of count ten, as shown in Figure 8–9(b).

▶ FIGURE 8–9



(a) 74LS93A connected as a modulus-16 counter

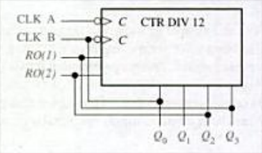
(b) 74LS93A connected as a decade counter

EXAMPLE 8–3

Show how the 74LS93A can be used as a modulus-12 counter.

Solution Use the gated reset inputs, *RO(1)* and *RO(2)*, to partially decode count 12 (remember, there is an internal NAND gate associated with these inputs). The count-12 decoding is accomplished by connecting Q_3 to *RO(1)* and Q_2 to *RO(2)*, as shown in Figure 8–10. Output Q_0 is connected to CLK B to create a 4-bit counter.

▶ FIGURE 8–10



Immediately after the counter goes to count 12 (1100), it is reset to 0000. The recycling, however, results in a glitch on Q_2 because the counter must go into the 1100 state for several nanoseconds before recycling.

Supplementary Problem Show how the 74LS93A can be connected as a modulus-13 counter.

SECTION 8–1 REVIEW

Answers are at the end of the chapter.

1. What does the term *asynchronous* mean in relation to counters?
2. How many states does a modulus-14 counter have? What is the minimum number of flip-flops required?

8–2 SYNCHRONOUS COUNTER OPERATION

The term *synchronous* refers to events that have a fixed time relationship with each other. With respect to counter operation, synchronous means that all the flip-flops in the counter are clocked at the same time by a common clock pulse.

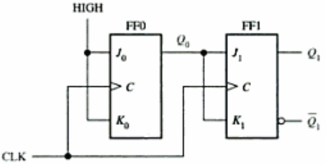
After completing this section, you should be able to

- Describe the operation of a 2-bit synchronous binary counter
- Describe the operation of a 3-bit synchronous binary counter
- Describe the operation of a 4-bit synchronous binary counter
- Describe the operation of a synchronous decade counter
- Develop counter timing diagrams
- Discuss the 74HC163 4-bit binary counter and the 74LS160 BCD decade counter

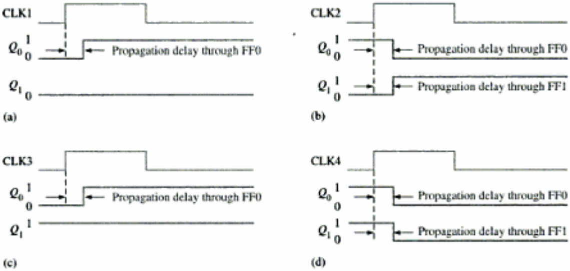
A 2-Bit Synchronous Binary Counter

Figure 8–11 shows a 2-bit **synchronous** binary counter. Notice that an arrangement different from that for the asynchronous counter must be used for the J_1 and K_1 inputs of FF1 in order to achieve a binary sequence.

▶ **FIGURE 8–11**


HIGH
FF0
FF1
Q_0
J_0
C
K_0
J_1
C
K_1
Q_1
$\overline{Q}_1$
CLK

The operation of this **synchronous counter** is as follows: First, assume that the counter is initially in the binary 0 state; that is, both flip-flops are RESET. When the positive edge of the first clock pulse is applied, FF0 will toggle and Q_0 will therefore go HIGH. What happens to FF1 at the positive-going edge of CLK1? To find out, let us look at the input conditions of FF1. Inputs J_1 and K_1 are both LOW because Q_0, to which they are connected, has not yet gone HIGH. Remember, there is a propagation delay from the triggering edge of the clock pulse until the Q output actually makes a transition. So, $J = 0$ and $K = 0$ when the leading edge of the first clock pulse is applied. This is a no-change condition, and therefore, FF1 does not change state. A timing detail of this portion of the counter operation is shown in Figure 8–12(a).


CLK1
Q_0 1 0
Propagation delay through FF0
Q_1 0
(a)
CLK2
Q_0 1 0
Propagation delay through FF0
Q_1 1 0
Propagation delay through FF1
(b)
CLK3
Q_0 1 0
Propagation delay through FF0
Q_1 1
(c)
CLK4
Q_0 1 0
Propagation delay through FF0
Q_1 1 0
Propagation delay through FF1
(d)

▲ **FIGURE 8–12**

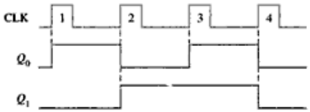
After CLK1, $Q_0 = 1$ and $Q_1 = 0$ (which is the binary 1 state). When the leading edge of CLK2 occurs, FF0 will toggle and Q_0 will go LOW. Since FF1 has a HIGH ($Q_0 = 1$) on its J_1 and K_1 inputs at the triggering edge of this clock pulse, the flip-flop toggles and Q_1 goes HIGH. Thus, after CLK2, $Q_0 = 0$ and $Q_1 = 1$ (which is a binary 2 state). The timing detail for this condition is shown in Figure 8–12(b).

When the leading edge of CLK3 occurs, FF0 again toggles to the SET state ($Q_0 = 1$), and FF1 remains SET ($Q_1 = 1$) because its J_1 and K_1 inputs are both LOW ($Q_0 = 0$). After this triggering edge, $Q_0 = 1$ and $Q_1 = 1$ (which is a binary 3 state). The timing detail is shown in Figure 8–12(c).

Finally, at the leading edge of CLK4, Q_0 and Q_1 go LOW because they both have a toggle condition on their J and K inputs. The timing detail is shown in Figure 8–12(d). The counter has now recycled to its original state, binary 0.

The complete timing diagram for the counter in Figure 8–11 is shown in Figure 8–13. Notice that all the waveform transitions appear coincident; that is, the propagation delays are not indicated. Although the delays are an important factor in the synchronous counter operation, in an overall timing diagram they are normally omitted for simplicity. Major waveform relationships resulting from the normal operation of a circuit can be conveyed completely without showing small delay and timing differences. However, in high-speed digital circuits, these small delays are an important consideration in design and troubleshooting.

FIGURE 8–13



A 3-Bit Synchronous Binary Counter

A 3-bit synchronous binary counter is shown in Figure 8–14, and its timing diagram is shown in Figure 8–15. You can understand this counter operation by examining its sequence of states as shown in Table 8–3.

FIGURE 8–14

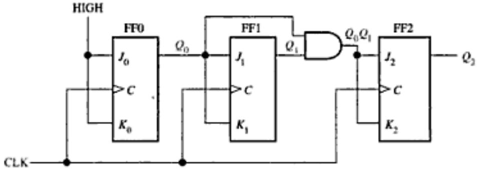


FIGURE 8–15

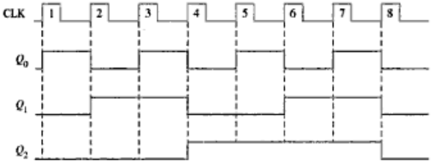


TABLE 8–3

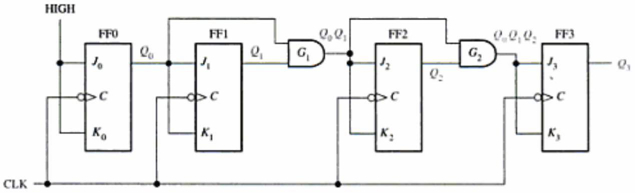
CLOCK PULSE	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

First, let us look at Q_0. Notice that Q_0 changes on each clock pulse as the counter progresses from its original state to its final state and then back to its original state. To produce this operation, FF0 must be held in the toggle mode by constant HIGHs on its J_0 and K_0 inputs. Notice that Q_1 goes to the opposite state following each time Q_0 is a 1. This change occurs at CLK2, CLK4, CLK6, and CLK8. The CLK8 pulse causes the counter to recycle. To produce this operation, Q_0 is connected to the J_1 and K_1 inputs of FF1. When Q_0 is a 1 and a clock pulse occurs, FF1 is in the toggle mode and therefore changes state. The other times, when Q_0 is a 0, FF1 is in the no-change mode and remains in its present state.

Next, let us see how FF2 is made to change at the proper times according to the binary sequence. Notice that both times Q_2 changes state, it is preceded by the unique condition in which both Q_0 and Q_1 are HIGH. This condition is detected by the AND gate and applied to the J_2 and K_2 inputs of FF2. Whenever both Q_0 and Q_1 are HIGH, the output of the AND gate makes the J_2 and K_2 inputs of FF2 HIGH, and FF2 toggles on the following clock pulse. At all other times, the J_2 and K_2 inputs of FF2 are held LOW by the AND gate output, and FF2 does not change state.

A 4-Bit Synchronous Binary Counter

Figure 8–16(a) shows a 4-bit synchronous binary counter, and Figure 8–16(b) shows its timing diagram. This particular counter is implemented with negative edge-triggered flip-flops. The reasoning behind the J and K input control for the first three flip-flops is the same as previously discussed for the 3-bit counter. The fourth stage, FF3, changes only twice in the sequence. Notice that both of these transitions occur following the times that Q_0, Q_1, and Q_2 are all HIGH. This condition is decoded by AND gate G_2 so that when a clock pulse occurs, FF3 will change state. For all other times the J_3 and K_3 inputs of FF3 are LOW, and it is in a no-change condition.



(a)

You can understand the counter operation by examining the sequence of states in Table 8–4 and by following the implementation in Figure 8–17. First, notice that FF0 (Q_0) toggles on each clock pulse, so the logic equation for its J_0 and K_0 inputs is

$$J_0 = K_0 = 1$$

This equation is implemented by connecting J_0 and K_0 to a constant HIGH level.

Next, notice in Table 8–4 that FF1 (Q_1) changes on the next clock pulse each time $Q_0 = 1$ and $Q_3 = 0$, so the logic equation for the J_1 and K_1 inputs is

$$J_1 = K_1 = Q_0\overline{Q}_3$$

This equation is implemented by ANDing Q_0 and $\overline{Q}_3$ and connecting the gate output to the J_1 and K_1 inputs of FF1.

▶ TABLE 8–4

CLOCK PULSE	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycles)	0	0	0	0

Flip-flop 2 (Q_2) changes on the next clock pulse each time both $Q_0 = 1$ and $Q_1 = 1$. This requires an input logic equation as follows:

$$J_2 = K_2 = Q_0Q_1$$

This equation is implemented by ANDing Q_0 and Q_1 and connecting the gate output to the J_2 and K_2 inputs of FF2.

Finally, FF3 (Q_3) changes to the opposite state on the next clock pulse each time $Q_0 = 1$, $Q_1 = 1$, and $Q_2 = 1$ (state 7), or when $Q_0 = 1$ and $Q_3 = 1$ (state 9). The equation for this is as follows:

$$J_3 = K_3 = Q_0Q_1Q_2 + Q_0Q_3$$

This function is implemented with the AND/OR logic connected to the J_3 and K_3 inputs of FF3 as shown in the logic diagram in Figure 8–17. Notice that the differences between this decade counter and the modulus-16 binary counter in Figure 8–16 are the $Q_0\overline{Q}_3$ AND gate, the Q_0Q_3 AND gate, and the OR gate; this arrangement detects the occurrence of the 1001 state and causes the counter to recycle properly on the next clock pulse.

COUNTERS

A 4-Bit Synchronous Binary Counter

The 74HC163 is an example of an integrated circuit 4-bit synchronous binary counter. A logic symbol is shown in Figure 8–19 with pin numbers in parentheses. This counter has several features in addition to the basic functions previously discussed for the general synchronous binary counter.

First, the counter can be synchronously preset to any 4-bit binary number by applying the proper levels to the parallel data inputs. When a LOW is applied to the $\overline{LOAD}$ input, the counter will assume the state of the data inputs on the next clock pulse. Thus, the counter sequence can be started with any 4-bit binary number.

Also, there is an active-LOW clear input ($\overline{CLR}$), which synchronously resets all four flip-flops in the counter. There are two enable inputs, *ENP* and *ENT*. These inputs must both be HIGH for the counter to sequence through its binary states. When at least one input is LOW, the counter is disabled. The ripple clock output (*RCO*) goes HIGH when the counter reaches a **terminal count** of fifteen ($TC = 15$). This output, in conjunction with the enable inputs, allows these counters to be cascaded for higher count sequences, as will be discussed later.

► FIGURE 8–19

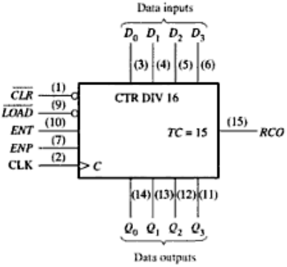


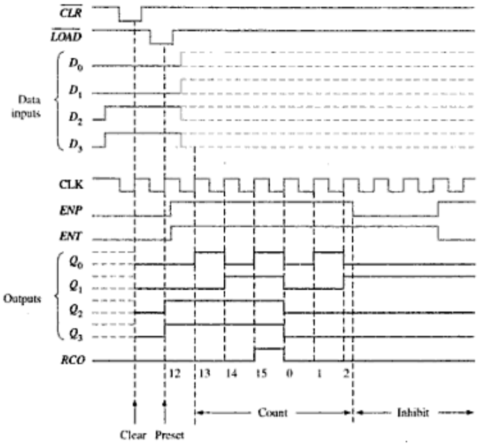
Figure 8–20 shows a timing diagram of this counter being preset to twelve (1100) and then counting up to its terminal count, fifteen (1111). Input D_0 is the least significant input bit, and Q_0 is the least significant output bit.

Let us examine this timing diagram in detail. This will aid you in interpreting timing diagrams found later in this chapter or on manufacturers' data sheets. To begin, the LOW level pulse on the $\overline{CLR}$ input causes all the outputs (Q_0, Q_1, Q_2, and Q_3) to go LOW.

Next, the LOW level pulse on the $\overline{LOAD}$ input synchronously enters the data on the data inputs (D_0, D_1, D_2, and D_3) into the counter. These data appear on the Q outputs at the time of the first positive-going clock edge after $\overline{LOAD}$ goes LOW. This is the preset operation. In this particular example, Q_0 is LOW, Q_1 is LOW, Q_2 is HIGH, and Q_3 is HIGH. This, of course, is a binary 12 (Q_0 is the LSB).

The counter now advances through states 13, 14, and 15 on the next three positive-going clock edges. It then recycles to 0, 1, 2 on the following clock pulses. Notice that both *ENP* and *ENT* inputs are HIGH during the state sequence. When *ENP* goes LOW, the counter is inhibited and remains in the binary 2 state.

FIGURE 8-20


CLR
LOAD
D0
D1
D2
D3
Data inputs
CLK
ENP
ENT
Q0
Q1
Q2
Q3
Outputs
RCO
12
13
14
15
0
1
2
Count
Inhibit
Clear
Preset

A Synchronous BCD Decade Counter

The 74LS160 is an example of a decade counter, which has the same inputs and outputs as the 74HC163 binary counter previously discussed. It can be preset to any BCD count by the use of the data inputs and a LOW on the $\overline{LOAD}$ input. A LOW on the asynchronous $\overline{CLR}$ will reset the counter. The enable inputs *ENP* and *ENT* must both be HIGH for the counter to advance through its sequence of states in response to a positive transition on the CLK input. As in the 74HC163, the enable inputs in conjunction with the ripple clock output *RCO* (terminal count of 1001) provide for cascading several decade counters. Figure 8-21 shows the logic symbol for the 74LS160 counter, and Figure 8-22 is a timing diagram showing the counter being preset to count 7 (0111). Cascaded counters will be discused in Section 8-5.

FIGURE 8-21

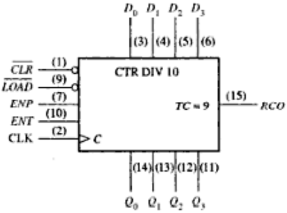
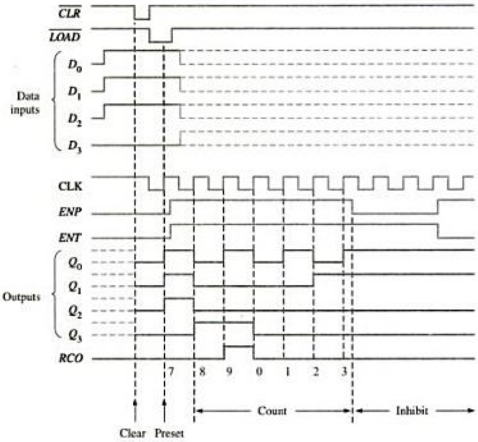

D0 D1 D2 D3
(3) (4) (5) (6)
CLR (1)
LOAD (9)
ENP (7)
ENT (10)
CLK (2)
CTR DIV 10
TC = 9
C
(15) RCO
(14) (13) (12) (11)
Q0 Q1 Q2 Q3

FIGURE 8–22


$\overline{CLR}$
$\overline{LOAD}$
Data inputs
D_0
D_1
D_2
D_3
CLK
ENP
ENT
Outputs
Q_0
Q_1
Q_2
Q_3
RCO
7
8
9
0
1
2
3
Count
Inhibit
Clear
Preset

SECTION 8–2 REVIEW

1. How does a synchronous counter differ from an asynchronous counter?
2. Explain the function of the preset feature of counters such as the 74LS160 and the 74HC163.
3. Describe the purpose of the *ENP* and *ENT* inputs and the *RCO* output for the two specific counters introduced in this section.

8–3 UP/DOWN SYNCHRONOUS COUNTERS

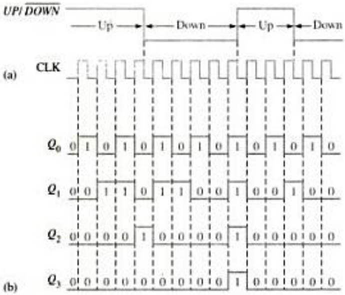
An up/down counter is one that is capable of progressing in either direction through a certain sequence. An up/down counter, sometimes called a bidirectional counter, can have any specified sequence of states. A 3-bit binary counter that advances upward through its sequence (0, 1, 2, 3, 4, 5, 6, 7) and then can be reversed so that it goes through the sequence in the opposite direction (7, 6, 5, 4, 3, 2, 1, 0) is an illustration of up/down sequential operation.

After completing this section, you should be able to

- Explain the basic operation of an up/down counter
- Discuss the 74HC190 up/down decade counter

EXAMPLE 8–4

Show the timing diagram and determine the sequence of a 4-bit synchronous binary up/down counter if the clock and $UP/\overline{DOWN}$ control inputs have waveforms as shown in Figure 8–24(a). The counter starts in the all 0s state and is positive edge-triggered.



▲ **FIGURE 8–24**

Solution The timing diagram showing the Q outputs is shown in Figure 8–24(b). From these waveforms, the counter sequence is as shown in Table 8–6.

▶ **TABLE 8–6**

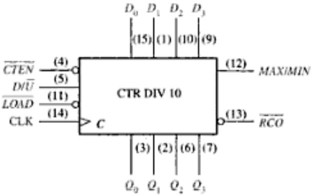
Q_3	Q_2	Q_1	Q_0	
0	0	0	0	UP
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	0	1	1	DOWN
0	0	1	0	
0	0	0	1	
0	0	0	0	
1	1	1	1	
0	0	0	0	UP
0	0	0	1	
0	0	1	0	
0	0	0	1	DOWN
0	0	0	0	

Supplementary Problem Show the timing diagram if the $UP/\overline{DOWN}$ control waveform in Figure 8–24(a) is inverted.

AN UP/DOWN DECADE COUNTER

Figure 8–25 shows a logic diagram for the 74HC190, an example of an integrated circuit up/down synchronous counter. The direction of the count is determined by the level of the up/down input ($D/\overline{U}$). When this input is HIGH, the counter counts down; when it is LOW, the counter counts up. Also, this device can be preset to any desired BCD digit as determined by the states of the data inputs when the $\overline{LOAD}$ input is LOW.

FIGURE 8–25

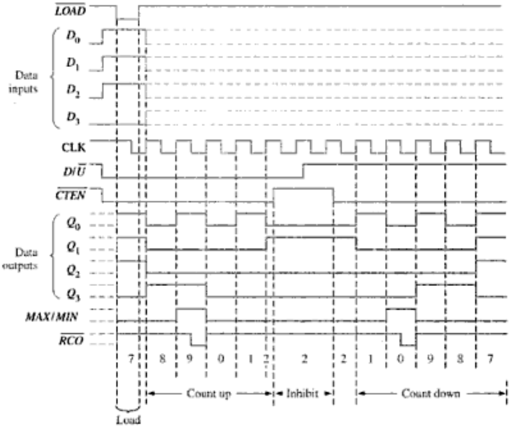


D_0 D_1 D_2 D_3
(15) (1) (10) (9)
$\overline{CTEN}$ (4)
$D/\overline{U}$ (5)
$\overline{LOAD}$ (11)
CLK (14)
CTR DIV 10
C
(12) MAX/MIN
(13) $\overline{RCO}$
(3) (2) (6) (7)
Q_0 Q_1 Q_2 Q_3

The *MAX/MIN* output produces a HIGH pulse when the terminal count nine (1001) is reached in the UP mode or when the terminal count zero (0000) is reached in the DOWN mode. This *MAX/MIN* output, along with the ripple clock output ($\overline{RCO}$) and the count enable input ($\overline{CTEN}$), is used when cascading counters. (Cascaded counters are discussed in Section 8–5.)

Figure 8–26 is an example timing diagram that shows the 74HC190 counter preset to seven (0111) and then going through a count-up sequence followed by a count-down sequence. The *MAX/MIN* output is HIGH when the counter is in either the all-0s state (*MIN*) or the 1001 state (*MAX*).

FIGURE 8–26



$\overline{LOAD}$
Data inputs
D_0
D_1
D_2
D_3
CLK
$D/\overline{U}$
$\overline{CTEN}$
Data outputs
Q_0
Q_1
Q_2
Q_3
MAX/MIN
$\overline{RCO}$
7 8 9 0 1 2 2 2 1 0 9 8 7
Count up
Inhibit
Count down
Load

SECTION 8-3 REVIEW

1. A 4-bit up/down binary counter is in the DOWN mode and in the 1010 state. On the next clock pulse, to what state does the counter go?
2. What is the terminal count of a 4-bit binary counter in the UP mode? In the DOWN mode? What is the next state after the terminal count in the DOWN mode?

8-4 DESIGN OF SYNCHRONOUS COUNTERS

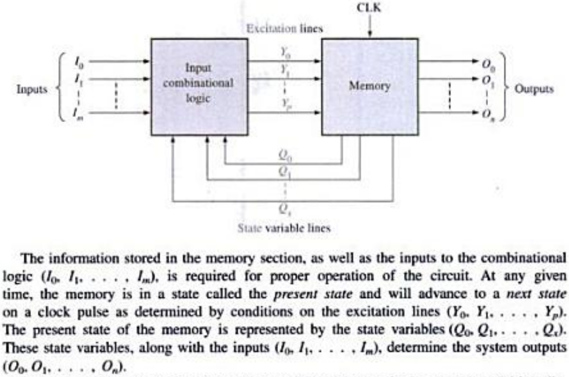
This section is recommended for those who want an introduction to counter design or to state machine design in general. It is not a prerequisite for any other material.

After completing this section, you should be able to

- Describe a general sequential circuit in terms of its basic parts and its input and outputs
- Develop a state diagram for a given sequence
- Develop a next-state table for a specified counter sequence
- Create a flip-flop transition table
- Use the Karnaugh map method to derive the logic requirements for a synchronous counter
- Implement a counter to produce a specified sequence of states

General Model of a Sequential Circuit

Before proceeding with a specific counter design technique, let us begin with a general definition of a **sequential circuit** or state machine: A general sequential circuit consists of a combinational logic section and a memory section (flip-flops), as shown in Figure 8–27. In a clocked sequential circuit, there is a clock input to the memory section as indicated.



► **FIGURE 8-27**

General clocked sequential circuit

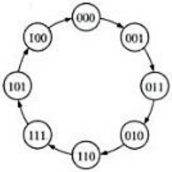
The information stored in the memory section, as well as the inputs to the combinational logic ($I_0, I_1, \ldots, I_m$), is required for proper operation of the circuit. At any given time, the memory is in a state called the *present state* and will advance to a *next state* on a clock pulse as determined by conditions on the excitation lines ($Y_0, Y_1, \ldots, Y_p$). The present state of the memory is represented by the state variables ($Q_0, Q_1, \ldots, Q_x$). These state variables, along with the inputs ($I_0, I_1, \ldots, I_m$), determine the system outputs ($O_0, O_1, \ldots, O_n$).

Not all sequential circuits have input and output variables as in the general model just discussed. However, all have excitation variables and state variables. Counters are a special case of clocked sequential circuits. In this section, a general design procedure for sequential circuits is applied to synchronous counters in a series of steps.

Step 1: State Diagram

A counter is first described by a **state diagram**, which shows the progression of states through which the counter advances when it is clocked. As an example, Figure 8–28 is a state diagram for a basic 3-bit Gray code counter. This particular circuit has no inputs other than the clock and no outputs other than the outputs taken off each flip-flop in the counter. You may wish to review the coverage of the Gray code in Chapter 2 at this time.

▶ **FIGURE 8–28**



Step 2: Next-state Table

Once the sequential circuit is defined by a state diagram, the second step is to derive a next-state table, which lists each state of the counter (present state) along with the corresponding next state. *The next state is the state that the counter goes to from its present state upon application of a clock pulse.* The next-state table is derived from the state diagram and is shown in Table 8–7 for the 3-bit Gray code counter. Q_0 is the least significant bit.

▶ **TABLE 8–7**

PRESENT STATE			NEXT STATE		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Step 3: Flip-flop Transition Table

Table 8–8 is a transition table for the J-K flip-flop. All possible output transitions are listed by showing the Q output of the flip-flop going from present states to next states. Q_N is the present state of the flip-flop (before a clock pulse) and Q_{N+1} is the next state (after a clock pulse). For each output transition, the J and K inputs that will cause the transition to occur are listed. The Xs indicate a "don't care" (the input can be either a 1 or a 0).

▶ **TABLE 8–8**

OUTPUT TRANSITIONS			FLIP-FLOP INPUTS	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

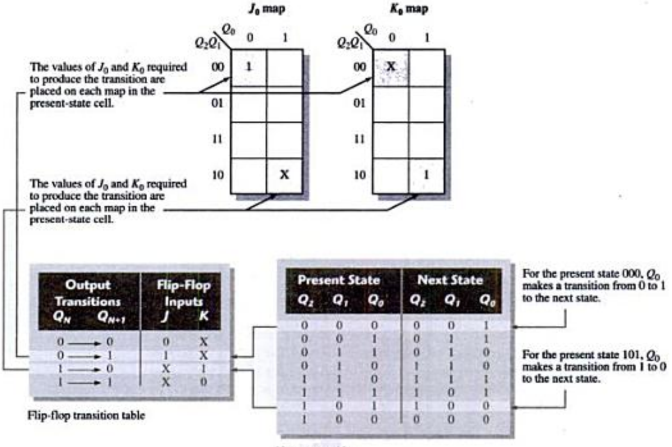
Q_N: present state
Q_{N+1}: next state
X: "don't care"

To design the counter, the transition table is applied to each of the flip-flops in the counter, based on the next-state table (Table 8–7). For example, for the present state 000, Q_0 goes from a present state of 0 to a next state of 1. To make this happen, J_0 must be a 1 and you don't care what K_0 is ($J_0 = 1$, $K_0 = X$), as you can see in the transition table (Table 8–8). Next, Q_1 is 0 in the present state and remains a 0 in the next state. For this transition, $J_1 = 0$ and $K_1 = X$. Finally, Q_2 is 0 in the present state and remains a 0 in the next state. Therefore, $J_2 = 0$ and $K_2 = X$. This analysis is repeated for each present state in Table 8–7.

Step 4: Karnaugh Maps

Karnaugh maps can be used to determine the logic required for the J and K inputs of each flip-flop in the counter. There is a Karnaugh map for the J input and a Karnaugh map for the K input of each flip-flop. In this design procedure, each cell in a Karnaugh map represents one of the present states in the counter sequence listed in Table 8–7.

From the J and K states in the transition table (Table 8–8) a 1, 0, or X is entered into each present state cell on the maps depending on the transition of the Q output for a particular flip-flop. To illustrate this procedure, two sample entries are shown for the J_0 and the K_0 inputs to the least significant flip-flop (Q_0) in Figure 8–29.



The values of J_0 and K_0 required to produce the transition are placed on each map in the present-state cell.

The values of J_0 and K_0 required to produce the transition are placed on each map in the present-state cell.

Output Transitions Q_N	Q_{N+1}	Flip-Flop Inputs J	K
0 ⟶	0	0	X
0 ⟶	1	1	X
1 ⟶	0	X	1
1 ⟶	1	X	0

Flip-flop transition table

Present State Q_2	Q_1	Q_0	Next State Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Next-state table

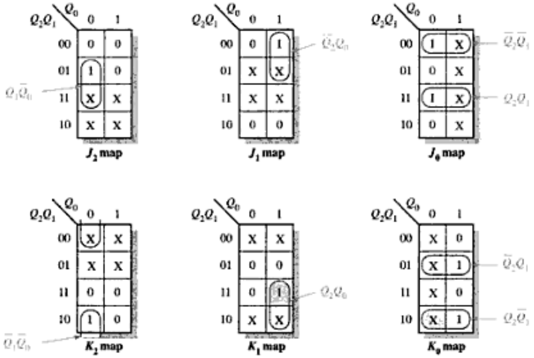
For the present state 000, Q_0 makes a transition from 0 to 1 to the next state.

For the present state 101, Q_0 makes a transition from 1 to 0 to the next state.

▲ FIGURE 8–29

The completed Karnaugh maps for all three flip-flops in the counter are shown in Figure 8–30. The cells are grouped as indicated and the corresponding Boolean expressions for each group are derived.

FIGURE 8–30


J_2 map
J_1 map
J_0 map
K_2 map
K_1 map
K_0 map
$Q_1\bar{Q}_0$
$\bar{Q}_2Q_0$
$\bar{Q}_2\bar{Q}_1$
Q_2Q_1
$\bar{Q}_1\bar{Q}_0$
Q_2Q_0
$\bar{Q}_2Q_1$
$Q_2\bar{Q}_1$

Step 5: Logic Expressions for Flip-flop Inputs

From the Karnaugh maps of Figure 8–30 you obtain the following expressions for the J and K inputs of each flip-flop:

$$J_0 = Q_2Q_1 + \bar{Q}_2\bar{Q}_1 = \overline{Q_2 \oplus Q_1}$$

$$K_0 = Q_2\bar{Q}_1 + \bar{Q}_2Q_1 = Q_2 \oplus Q_1$$

$$J_1 = \bar{Q}_2Q_0$$

$$K_1 = Q_2Q_0$$

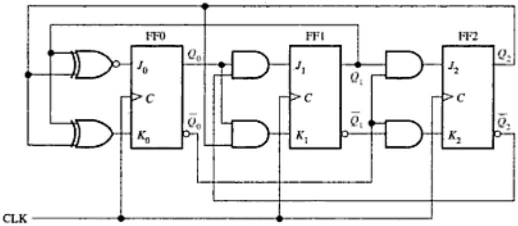
$$J_2 = Q_1\bar{Q}_0$$

$$K_2 = \bar{Q}_1\bar{Q}_0$$

Step 6: Counter Implementation

The final step is to implement the combinational logic from the expressions for the J and K inputs and connect the flip-flops to form the complete 3-bit Gray code counter as shown in Figure 8–31.

FIGURE 8–31


FF0
FF1
FF2
J_0
K_0
J_1
K_1
J_2
K_2
C
Q_0
$\bar{Q}_0$
Q_1
$\bar{Q}_1$
Q_2
$\bar{Q}_2$
CLK

A summary of steps used in the design of this counter follows. In general, these steps can be applied to any sequential circuit.

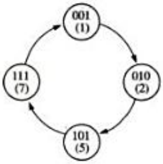
1. Specify the counter sequence and draw a state diagram.
2. Derive a next-state table from the state diagram.
3. Develop a transition table showing the flip-flop inputs required for each transition. The transition table is always the same for a given type of flip-flop.
4. Transfer the *J* and *K* states from the transition table to Karnaugh maps. There is a Karnaugh map for each input of each flip-flop.
5. Group the Karnaugh map cells to generate and derive the logic expression for each flip-flop input.
6. Implement the expressions with combinational logic, and combine with the flip-flops to create the counter.

This procedure is now applied to the design of other synchronous counters in Examples 8–5 and 8–6.

EXAMPLE 8–5

Design a counter with the irregular binary count sequence shown in the state diagram of Figure 8–32. Use J-K flip-flops.

FIGURE 8–32



Solution **Step 1.** The state diagram is as shown. Although there are only four states, a 3-bit counter is required to implement this sequence because the maximum binary count is seven. Since the required sequence does not include all the possible binary states, the invalid states (0, 3, 4, and 6) can be treated as "don't cares" in the design. However, if the counter should erroneously get into an invalid state, you must make sure that it goes back to a valid state.

Step 2. The next-state table is developed from the state diagram and is given in Table 8–9.

TABLE 8–9

PRESENT STATE			NEXT STATE		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	1	0	1	0
0	1	0	1	0	1
1	0	1	1	1	1
1	1	1	0	0	1

Step 3. The transition table for the J-K flip-flop is repeated in Table 8–10.

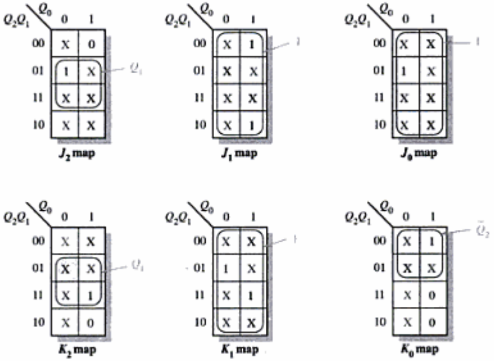
▶ TABLE 8–10

Transition table for a J-K flip-flop

OUTPUT TRANSITIONS			FLIP-FLOP INPUTS	
Q_N		Q_{N+1}	J	K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Step 4. The J and K inputs are plotted on the present-state Karnaugh maps in Figure 8–33. Also "don't cares" can be placed in the cells corresponding to the invalid states of 000, 011, 100, and 110, as indicated by the red Xs.

▶ FIGURE 8–33

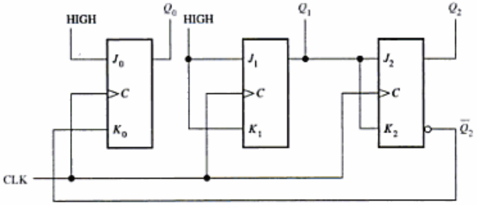

Q_2Q_1
Q_0
Q_1
1
1
J_2 map
J_1 map
J_0 map
Q_1
1
$\overline{Q}_2$
K_2 map
K_1 map
K_0 map

Step 5. Group the 1s, taking advantage of as many of the "don't care" states as possible for maximum simplification, as shown in Figure 8–33. Notice that when *all* cells in a map are grouped, the expression is simply equal to 1. The expression for each J and K input taken from the maps is as follows:

$$J_0 = 1, K_0 = \overline{Q}_2$$
$$J_1 = K_1 = 1$$
$$J_2 = K_2 = Q_1$$

Step 6. The implementation of the counter is shown in Figure 8–34.

▶ FIGURE 8–34


HIGH
Q_0
HIGH
Q_1
Q_2
J_0
C
K_0
J_1
C
K_1
J_2
C
K_2
$\overline{Q}_2$
CLK

An analysis shows that if the counter, by accident, gets into one of the invalid states (0, 3, 4, 6), it will always return to a valid state according to the following sequences: 0 → 3 → 4 → 7, and 6 → 1.

Supplementary Problem Verify the analysis that proves the counter will always return (eventually) to a valid state from an invalid state.

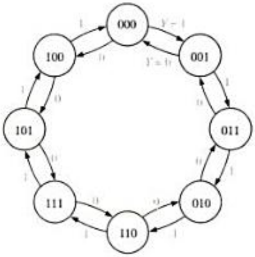
EXAMPLE 8–6

Develop a synchronous 3-bit up/down counter with a Gray code sequence. The counter should count up when an UP/$\overline{\text{DOWN}}$ control input is 1 and count down when the control input is 0.

Solution **Step 1.** The state diagram is shown in Figure 8–35. The 1 or 0 beside each arrow indicates the state of the UP/$\overline{\text{DOWN}}$ control input, Y.

FIGURE 8–35

State diagram for a 3-bit up/down Gray code counter



Step 2. The next-state table is derived from the state diagram and is shown in Table 8–11. Notice that for each present state there are two possible next states, depending on the UP/$\overline{\text{DOWN}}$ control variable, Y.

TABLE 8–11

Next-state table for 3-bit up/down Gray code counter

PRESENT STATE			NEXT STATE					
			$Y = 0$ (DOWN)			$Y = 1$ (UP)		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	1	0
0	1	0	0	1	1	1	1	0
1	1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0	1
1	0	1	1	1	1	1	0	0
1	0	0	1	0	1	0	0	0

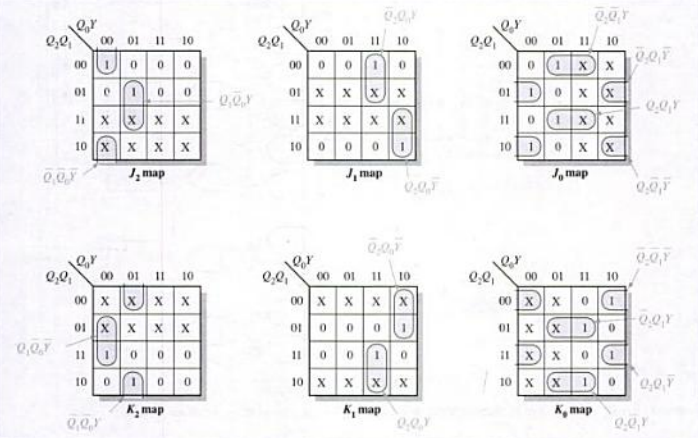
Y = UP/$\overline{\text{DOWN}}$ control input.

Step 3. The transition table for the J-K flip-flops is repeated in Table 8–12.

TABLE 8–12

OUTPUT TRANSITIONS Q_N		OUTPUT TRANSITIONS Q_{N+1}	FLIP-FLOP INPUTS J	FLIP-FLOP INPUTS K
0	⟶	0	0	X
0	⟶	1	1	X
1	⟶	0	X	1
1	⟶	1	X	0

Step 4. The Karnaugh maps for the J and K inputs of the flip-flops are shown in Figure 8–36. The UP/$\overline{\text{DOWN}}$ control input, Y, is considered one of the state variables along with Q_0, Q_1, and Q_2. Using the next-state table, the information in the "Flip-Flop Inputs" column of Table 8–12 is transferred onto the maps as indicated for each present state of the counter.



Q_2Q_1
Q_0Y
00
01
11
10
J_2 map
J_1 map
J_0 map
K_2 map
K_1 map
K_0 map
$Q_1\overline{Q}_0Y$
$\overline{Q}_1\overline{Q}_0\overline{Y}$
$\overline{Q}_2Q_0Y$
$Q_2Q_0\overline{Y}$
$\overline{Q}_2\overline{Q}_1Y$
$\overline{Q}_2Q_1\overline{Y}$
Q_2Q_1Y
$Q_2\overline{Q}_1\overline{Y}$
$Q_1\overline{Q}_0\overline{Y}$
$\overline{Q}_1\overline{Q}_0Y$
$\overline{Q}_2Q_0\overline{Y}$
Q_2Q_0Y
$\overline{Q}_2\overline{Q}_1\overline{Y}$
$\overline{Q}_2Q_1Y$
$Q_2Q_1\overline{Y}$
$Q_2\overline{Q}_1Y$

▲ **FIGURE 8–36**

Step 5. The 1s are combined in the largest possible groupings, with "don't cares" (Xs) used where possible. The groups are factored, and the expressions for the J and K inputs are as follows:

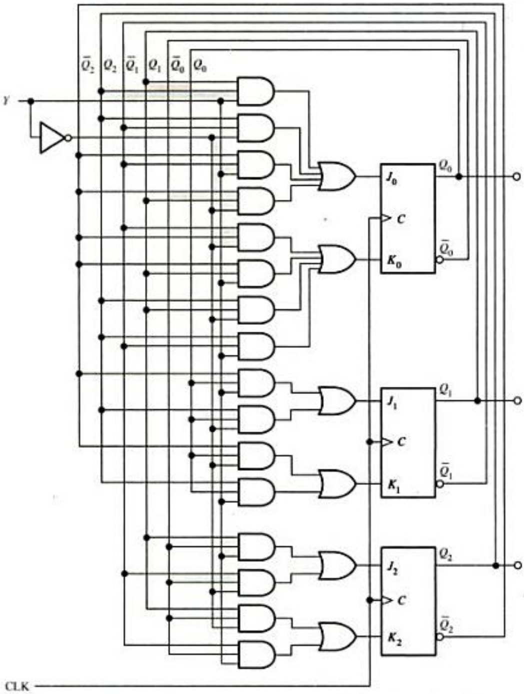
$$J_0 = Q_2Q_1Y + Q_2\overline{Q}_1\overline{Y} + \overline{Q}_2\overline{Q}_1Y + \overline{Q}_2Q_1\overline{Y} \qquad K_0 = \overline{Q}_2\overline{Q}_1\overline{Y} + \overline{Q}_2Q_1Y + Q_2\overline{Q}_1Y + Q_2Q_1\overline{Y}$$

$$J_1 = \overline{Q}_2Q_0Y + Q_2Q_0\overline{Y} \qquad K_1 = \overline{Q}_2Q_0\overline{Y} + Q_2Q_0Y$$

$$J_2 = Q_1\overline{Q}_0Y + \overline{Q}_1\overline{Q}_0\overline{Y} \qquad K_2 = Q_1\overline{Q}_0\overline{Y} + \overline{Q}_1\overline{Q}_0Y$$

Step 6. The J and K equations are implemented with combinational logic, and the complete counter is shown in Figure 8–37.

FIGURE 8–37

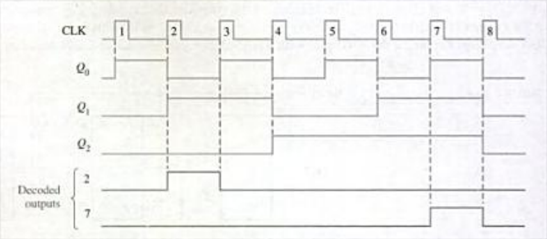


Supplementary Problem Verify that the logic in Figure 8–37 agrees with the expressions in Step 5.

In this section, you have seen how sequential circuit design techniques can be applied specifically to counter design. In general, sequential circuits can be classified into two types: (1) those in which the output or outputs depend only on the present internal state (called *Moore circuits*) and (2) those in which the output or outputs depend on both the present state and the input or inputs (called *Mealy circuits*).

SECTION 8–4 REVIEW

1. A flip-flop is presently in the RESET state and must go to the SET state on the next clock pulse. What must *J* and *K* be?
2. A flip-flop is presently in the SET state and must remain SET on the next clock pulse. What must *J* and *K* be?



▲ FIGURE 8–46

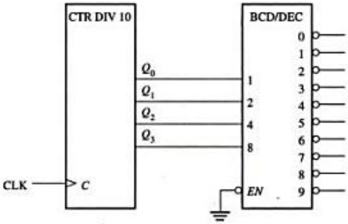
Supplementary Problem Show the logic for decoding state 5 in the 3-bit counter.

Decoding Glitches

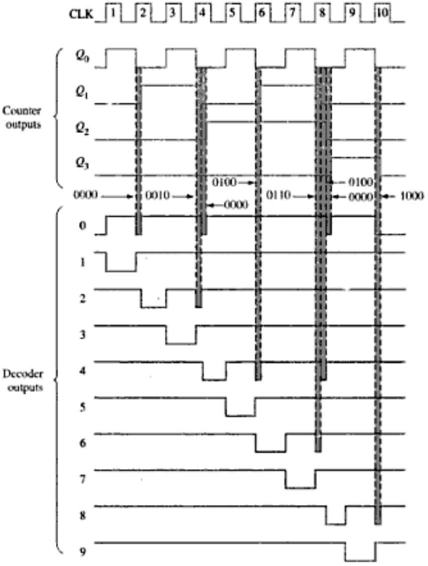
The problem of glitches produced by the decoding process was introduced in Chapter 6. As you have learned, the propagation delays due to the ripple effect in asynchronous counters create transitional states in which the counter outputs are changing at slightly different times. These transitional states produce undesired voltage spikes of short duration (glitches) on the outputs of a decoder connected to the counter. The glitch problem can also occur to some degree with synchronous counters because the propagation delays from clock to Q outputs of each flip-flop in a counter can vary slightly.

Figure 8–47 shows a basic asynchronous BCD decade counter connected to a BCD-to-decimal decoder. To see what happens in this case, let us look at a timing diagram in which the propagation delays are taken into account, as shown in Figure 8–48. Notice that these delays cause false states of short duration. The value of the false binary state at each critical transition is indicated on the diagram. The resulting glitches can be seen on the decoder outputs.

▶ FIGURE 8–47



▶ FIGURE 8–48


CLK
1
2
3
4
5
6
7
8
9
10
Counter outputs
Q_0
Q_1
Q_2
Q_3
0000
0010
0100
0000
0110
0100
0000
1000
Decoder outputs
0
1
2
3
4
5
6
7
8
9

One way to eliminate the glitches is to enable the decoded outputs at a time after the glitches have had time to disappear. This method is known as *strobing* and can be accomplished in the case of an active-HIGH clock by using the LOW level of the clock to enable the decoder, as shown in Figure 8–49. The resulting improved timing diagram is shown in Figure 8–50.

▶ FIGURE 8–49

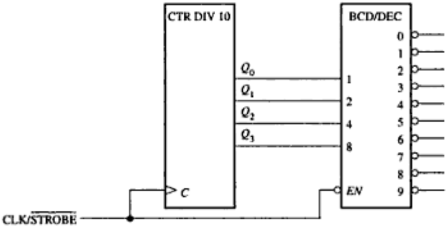
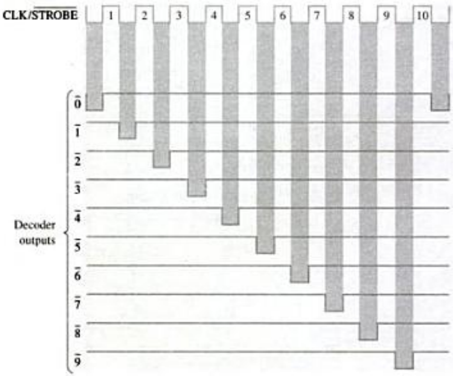

CTR DIV 10
Q_0
Q_1
Q_2
Q_3
C
BCD/DEC
1
2
4
8
EN
0
1
2
3
4
5
6
7
8
9
CLK/$\overline{\text{STROBE}}$

FIGURE 8–50



SECTION 8–6 REVIEW

1. What transitional states are possible when a 4-bit asynchronous binary counter changes from
   - (a) count 2 to count 3
   - (b) count 3 to count 4
   - (c) count 10_{10} to count 11_{10}
   - (d) count 15 to count 0

8–7 COUNTER APPLICATIONS

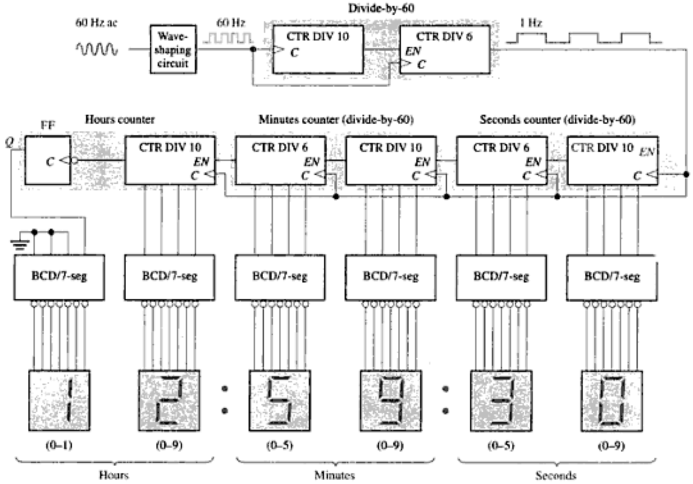
The digital counter is a useful and versatile device that is found in many applications. In this section, some representative counter applications are presented.

After completing this section, you should be able to

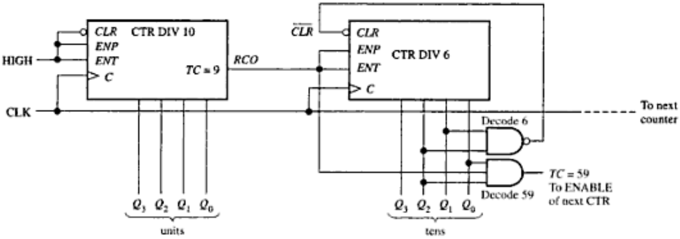
- Describe how counters are used in a basic digital clock system
- Explain how a divide-by-60 counter is implemented and how it is used in a digital clock
- Explain how the hours counter is implemented
- Discuss the application of a counter in an automobile parking control system
- Describe how a counter is used in the process of parallel-to-serial data conversion

Application Examples

A Digital Clock A common example of a counter application is in timekeeping systems. Figure 8–51 is a simplified logic diagram of a digital clock that displays seconds, minutes, and hours. First, a 60 Hz sinusoidal ac voltage is converted to a 60 Hz pulse waveform and divided down to a 1 Hz pulse waveform by a divide-by-60 counter formed by a divide-by-10 counter followed by a divide-by-6 counter. Both the *seconds* and *minutes* counts are also produced by divide-by-60 counters, the details of which are shown in Figure 8–52. These counters count from 0 to 59 and then recycle to 0; synchronous decade counters are used in this particular implementation. Notice that the divide-by-6 portion is formed with a decade counter with a truncated sequence achieved by using the decoder count 6 to asynchronously clear the counter. The terminal count, 59, is also decoded to enable the next counter in the chain.


Divide-by-60
60 Hz ac
Wave-shaping circuit
60 Hz
CTR DIV 10
C
CTR DIV 6
EN
C
1 Hz
FF
Q
C
Hours counter
Minutes counter (divide-by-60)
Seconds counter (divide-by-60)
CTR DIV 10
EN
C
CTR DIV 6
CTR DIV 10
CTR DIV 6
CTR DIV 10
BCD/7-seg
(0–1)
(0–9)
(0–5)
(0–9)
(0–5)
(0–9)
Hours
Minutes
Seconds

▲ **FIGURE 8–51**


HIGH
CLK
CLR
ENP
ENT
C
CTR DIV 10
TC = 9
RCO
$\overline{CLR}$
CTR DIV 6
Decode 6
Decode 59
TC = 59
To ENABLE of next CTR
To next counter
Q_3 Q_2 Q_1 Q_0
units
tens

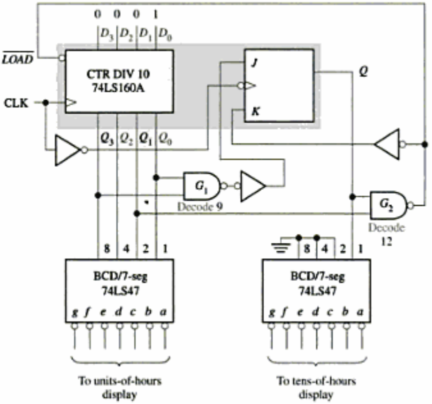
▲ **FIGURE 8–52**

Logic diagram of typical divide-by-60 counter using 74LS160A

The *hours* counter is implemented with a decade counter and a flip-flop as shown in Figure 8–53. Consider that initially both the decade counter and the flip-flop are RESET, and the decode-12 gate and decode-9 gate outputs are HIGH. The decade counter advances through all of its states from zero to nine, and on the clock pulse that recycles it from nine back to zero, the flip-flop goes to the SET state ($J = 1$, $K = 0$). This illuminates a 1 on the tens-of-hours display. The total count is now ten (the decade counter is in the zero state and the flip-flop is SET).

Next, the total count advances to eleven and then to twelve. In state 12 the Q_2 output of the decade counter is HIGH, the flip-flop is still SET, and thus the decode-12 gate output is LOW. This activates the $\overline{LOAD}$ input of the decade counter. On the next clock pulse, the decade counter is preset to state 1 by the data inputs, and the flip-flop is RESET ($J = 0$, $K = 1$). As you can see, this logic always causes the counter to recycle from twelve back to one rather than back to zero.

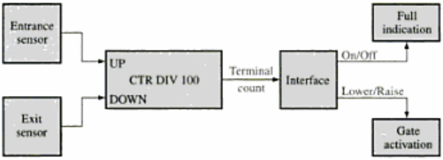
▶ **FIGURE 8–53**


0 0 0 1
D_3 D_2 D_1 D_0
$\overline{LOAD}$
CLK
CTR DIV 10
74LS160A
J
K
Q
Q_3 Q_2 Q_1 Q_0
G_1
Decode 9
G_2
Decode 12
8 4 2 1
8 4 2 1
BCD/7-seg
74LS47
g f e d c b a
BCD/7-seg
74LS47
g f e d c b a
To units-of-hours display
To tens-of-hours display

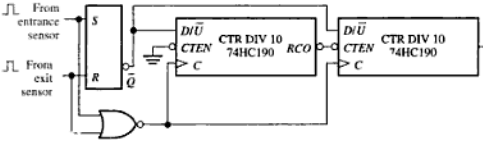
Automobile Parking Control A simple application example illustrates the use of an up/down counter to solve an everyday problem. The problem is to devise a means of monitoring available spaces in a one-hundred-space parking garage and provide for an indication of a full condition by illuminating a display sign and lowering a gate bar at the entrance.

A system that solves this problem consists of (1) optoelectronic sensors at the entrance and exit of the garage, (2) an up/down counter and associated circuitry, and (3) an interface circuit that uses the counter output to turn the FULL sign on or off as required and lower or raise the gate bar at the entrance. A general block diagram of this system is shown in Figure 8–54.

▶ **FIGURE 8–54**


Entrance sensor
Exit sensor
UP
CTR DIV 100
DOWN
Terminal count
Interface
On/Off
Lower/Raise
Full indication
Gate activation

A logic diagram of the up/down counter is shown in Figure 8–55. It consists of two cascaded 74HC190 up/down decade counters. The operation is described in the following paragraphs.


From entrance sensor
From exit sensor
S
R
$\overline{Q}$
$D/\overline{U}$
$\overline{CTEN}$
C
CTR DIV 10
74HC190
$\overline{RCO}$
$D/\overline{U}$
$\overline{CTEN}$
C
CTR DIV 10
74HC190

MAX/MIN (to interface) HIGH activates FULL sign and lowers gate.

▲ **FIGURE 8–55**

The counter is initially preset to 0 using the parallel data inputs, which are not shown. Each automobile entering the garage breaks a light beam, activating a sensor that produces an electrical pulse. This positive pulse sets the S-R latch on its leading edge. The LOW on the $\overline{Q}$ output of the latch puts the counter in the UP mode. Also, the sensor pulse goes through the NOR gate and clocks the counter on the LOW-to-HIGH transition of its trailing edge. Each time an automobile enters the garage, the counter is advanced by one **(incremented)**. When the one-hundredth automobile enters, the counter goes to its last state (100_{10}). The *MAX/MIN* output goes HIGH and activates the interface circuit (no detail), which lights the FULL sign and lowers the gate bar to prevent further entry.

When an automobile exits, an optoelectronic sensor produces a positive pulse, which resets the S-R latch and puts the counter in the DOWN mode. The trailing edge of the clock decreases the count by one **(decremented)**. If the garage is full and an automobile leaves, the *MAX/MIN* output of the counter goes LOW, turning off the FULL sign and raising the gate.

Parallel-to-Serial Data Conversion (Multiplexing) A simplified example of data transmission using multiplexing and demultiplexing techniques was introduced in Chapter 6. Essentially, the parallel data bits on the multiplexer inputs are converted to serial data bits on the single transmission line. A group of bits appearing simultaneously on parallel lines is called *parallel data*. A group of bits appearing on a single line in a time sequence is called *serial data.*

Parallel-to-serial conversion is normally accomplished by the use of a counter to provide a binary sequence for the data-select inputs of a data selector/multiplexer, as illustrated in Figure 8–56. The Q outputs of the modulus-8 counter are connected to the data-select inputs of an 8-bit multiplexer.

► **FIGURE 8–56**

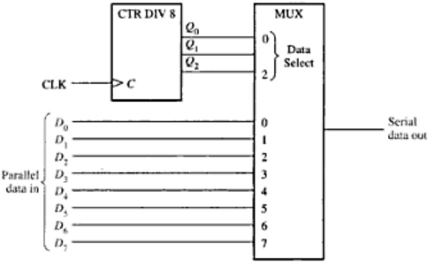
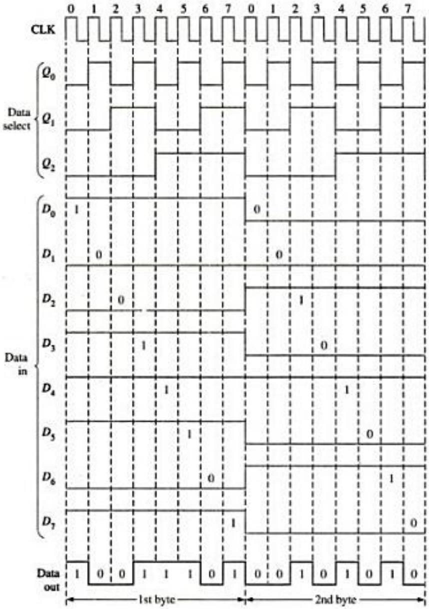

CTR DIV 8
Q_0
Q_1
Q_2
CLK
C
MUX
0
2
Data Select
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
Parallel data in
0
1
2
3
4
5
6
7
Serial data out

Figure 8–57 is a timing diagram illustrating the operation of this circuit. The first byte (eight-bit group) of parallel data is applied to the multiplexer inputs. As the counter goes through a binary sequence from zero to seven, each bit, beginning with D_0, is sequentially selected and passed through the multiplexer to the output line. After eight clock pulses the data byte has been converted to a serial format and sent out on the transmission line. When the counter recycles back to 0, the next byte is applied to the data inputs and is sequentially converted to serial form as the counter cycles through its eight states. This process continues repeatedly as each parallel byte is converted to a serial byte.

▶ FIGURE 8–57


0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
CLK
Data select
Q_0
Q_1
Q_2
Data in
D_0 1 0
D_1 0 0
D_2 0 1
D_3 1 0
D_4 1 1
D_5 1 0
D_6 0 1
D_7 1 0
Data out
1 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0
1st byte
2nd byte

SECTION 8–7 REVIEW

1. Explain the purpose of each NAND gate in Figure 8–53.
2. Identify the two recycle conditions for the hours counter in Figure 8–51, and explain the reason for each.

SELF-TEST

Answers are at the end of the chapter.

1. Asynchronous counters are known as
   (a) ripple counters (b) multiple clock counters
   (c) decade counters (d) modulus counters
2. An asynchronous counter differs from a synchronous counter in
   (a) the number of states in its sequence (b) the method of clocking
   (c) the type of flip-flops used (d) the value of the modulus
3. The modulus of a counter is
   (a) the number of flip-flops
   (b) the actual number of states in its sequence
   (c) the number of times it recycles in a second
   (d) the maximum possible number of states
4. A 3-bit binary counter has a maximum modulus of
   (a) 3 (b) 6 (c) 8 (d) 16
5. A 4-bit binary counter has a maximum modulus of
   (a) 16 (b) 32 (c) 8 (d) 4
6. A modulus-12 counter must have
   (a) 12 flip-flops (b) 3 flip-flops
   (c) 4 flip-flops (d) synchronous clocking
7. Which one of the following is an example of a counter with a truncated modulus?
   (a) Modulus 8 (b) Modulus 14
   (c) Modulus 16 (d) Modulus 32
8. A 4-bit ripple counter consists of flip-flops that each have a propagation delay from clock to Q output of 12 ns. For the counter to recycle from 1111 to 0000, it takes a total of
   (a) 12 ns (b) 24 ns (c) 48 ns (d) 36 ns
9. A BCD counter is an example of
   (a) a full-modulus counter (b) a decade counter
   (c) a truncated-modulus counter (d) answers (b) and (c)
10. Which of the following is an invalid state in an 8421 BCD counter?
    (a) 1100 (b) 0010 (c) 0101 (d) 1000
11. Three cascaded modulus-10 counters have an overall modulus of
    (a) 30 (b) 100 (c) 1000 (d) 10,000
12. A 10 MHz clock frequency is applied to a cascaded counter consisting of a modulus-5 counter, a modulus-8 counter, and two modulus-10 counters. The lowest output frequency possible is
    (a) 10 kHz (b) 2.5 kHz (c) 5 kHz (d) 25 kHz
13. A 4-bit binary up/down counter is in the binary state of zero. The next state in the DOWN mode is
    (a) 0001 (b) 1111 (c) 1000 (d) 1110
14. The terminal count of a modulus-13 binary counter is
    (a) 0000 (b) 1111 (c) 1101 (d) 1100

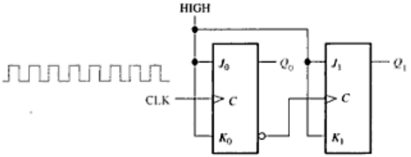
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 8–1 Asynchronous Counter Operation

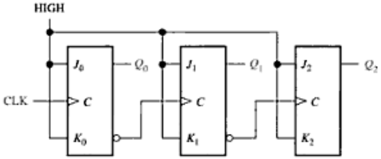
1. For the ripple counter shown in Figure 8–60, show the complete timing diagram for eight clock pulses, showing the clock, Q_0, and Q_1 waveforms.

▶ **FIGURE 8–60**



2. For the ripple counter in Figure 8–61, show the complete timing diagram for sixteen clock pulses. Show the clock, Q_0, Q_1, and Q_2 waveforms.

▶ **FIGURE 8–61**



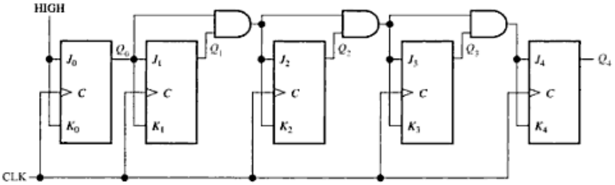
3. In the counter of Problem 2, assume that each flip-flop has a propagation delay from the triggering edge of the clock to a change in the Q output of 8 ns. Determine the worst-case (longest) delay time from a clock pulse to the arrival of the counter in a given state. Specify the state or states for which this worst-case delay occurs.

4. Show how to connect a 74LS93A 4-bit asynchronous counter for each of the following moduli:
   (a) 9 **(b)** 11 **(c)** 13 **(d)** 14 **(e)** 15

SECTION 8–2 Synchronous Counter Operation

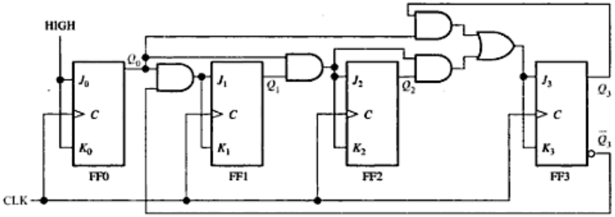
5. If the counter of Problem 3 were synchronous rather than asynchronous, what would be the longest delay time?

6. Show the complete timing diagram for the 5-stage synchronous binary counter in Figure 8–62. Verify that the waveforms of the Q outputs represent the proper binary number after each clock pulse.



▲ **FIGURE 8–62**

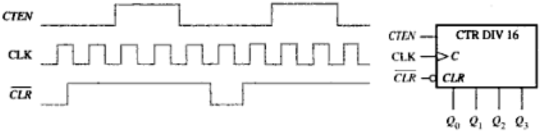
7. By analyzing the J and K inputs to each flip-flop prior to each clock pulse, prove that the decade counter in Figure 8-63 progresses through a BCD sequence. Explain how these conditions in each case cause the counter to go to the next proper state.


HIGH
Q_0
J_0
C
K_0
FF0
J_1
C
K_1
Q_1
FF1
J_2
C
K_2
Q_2
FF2
J_3
C
K_3
Q_3
$\overline{Q}_3$
FF3
CLK

▲ **FIGURE 8-63**

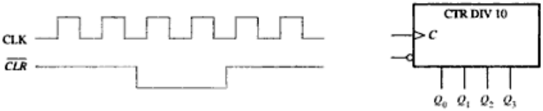
8. The waveforms in Figure 8-64 are applied to the count enable, clear, and clock inputs as indicated. Show the counter output waveforms in proper relation to these inputs. The clear input is asynchronous.

FIGURE 8-64


CTEN
CLK
$\overline{CLR}$
CTEN
CTR DIV 16
CLK
C
$\overline{CLR}$
CLR
Q_0 Q_1 Q_2 Q_3

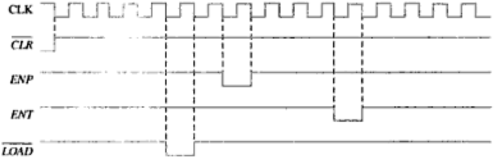
9. A BCD decade counter is shown in Figure 8-65. The waveforms are applied to the clock and clear inputs as indicated. Determine the waveforms for each of the counter outputs (Q_0, Q_1, Q_2, and Q_3). The clear is synchronous, and the counter is initially in the binary 1000 state.

FIGURE 8-65


CLK
$\overline{CLR}$
CTR DIV 10
C
Q_0 Q_1 Q_2 Q_3

10. The waveforms in Figure 8-66 are applied to a 74HC163 counter. Determine the Q outputs and the RCO. The inputs are $D_0 = 1$, $D_1 = 1$, $D_2 = 0$, and $D_3 = 1$.

FIGURE 8-66


CLK
$\overline{CLR}$
ENP
ENT
$\overline{LOAD}$

11. The waveforms in Figure 8-66 are applied to a 74LS160 counter. Determine the Q outputs and the RCO. The inputs are $D_0 = 1$, $D_1 = 0$, $D_2 = 0$, and $D_3 = 1$.

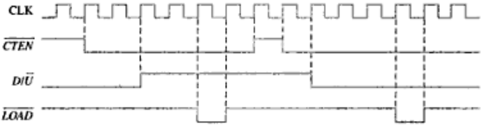
SECTION 8–3 Up/Down Synchronous Counters

12. Show a complete timing diagram for a 3-bit up/down counter that goes through the following sequence. Indicate when the counter is in the UP mode and when it is in the DOWN mode. Assume positive edge-triggering.

0, 1, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0

13. Develop the Q output waveforms for a 74HC190 up/down counter with the input waveforms shown in Figure 8–67. A binary 0 is on the data inputs. Start with a count of 0000.

▶ **FIGURE 8–67**



SECTION 8–4 Design of Synchronous Counters

14. Determine the sequence of the counter in Figure 8–68.

15. Determine the sequence of the counter in Figure 8–69. Begin with the counter cleared.

16. Design a counter to produce the following sequence. Use J-K flip-flops.

00, 10, 01, 11, 00, …

17. Design a counter to produce the following binary sequence. Use J-K flip-flops.

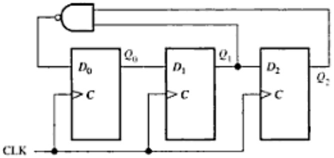
1, 4, 3, 5, 7, 6, 2, 1, …

18. Design a counter to produce the following binary sequence. Use J-K flip-flops.

0, 9, 1, 8, 2, 7, 3, 6, 4, 5, 0, …

19. Design a binary counter with the sequence shown in the state diagram of Figure 8–70.

▶ **FIGURE 8–68**



▶ **FIGURE 8–69**

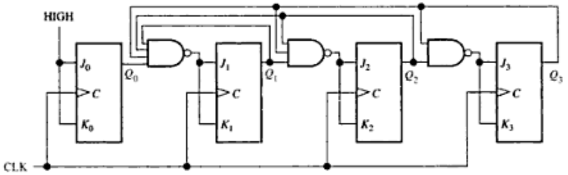
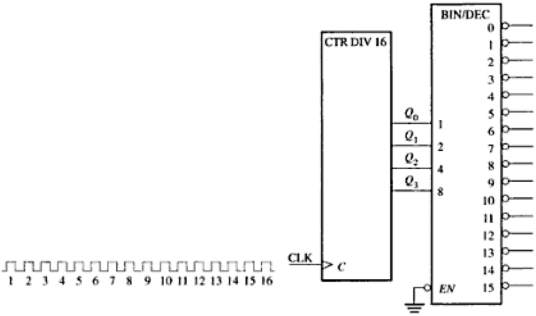


FIGURE 8–72



25. If the counter in Figure 8–72 is asynchronous, determine where the decoding glitches occur on the decoder output waveforms.

26. Modify the circuit in Figure 8–72 to eliminate decoding glitches.

27. Analyze the counter in Figure 8–45 for the occurrence of glitches on the decode gate output. If glitches occur, suggest a way to eliminate them.

28. Analyze the counter in Figure 8–46 for the occurrence of glitches on the outputs of the decoding gates. If glitches occur, make a design change that will eliminate them.

SECTION 8–7 Counter Applications

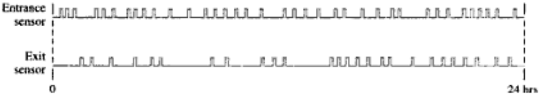
29. Assume that the digital clock of Figure 8–51 is initially reset to 12 o'clock. Determine the binary state of each counter after sixty-two 60 Hz pulses have occurred.

30. What is the output frequency of each counter in the digital clock circuit of Figure 8–51?

31. For the automobile parking control system in Figure 8–54, a pattern of entrance and exit sensor pulses during a given 24-hour period are shown in Figure 8–73. If there were 53 cars already in the garage at the beginning of the period, what is the state of the counter at the end of the 24 hours?

32. The binary number for decimal 57 appears on the parallel data inputs of the parallel-to-serial converter in Figure 8–56 (D_0 is the LSB). The counter initially contains all zeros and a 10 kHz clock is applied. Develop the timing diagram showing the clock, the counter outputs, and the serial data output.

FIGURE 8–73



ANSWERS

SECTION REVIEWS

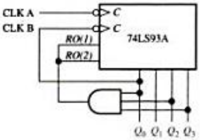
SECTION 8–1 Asynchronous Counter Operation

1. Asynchronous means that each flip-flop after the first one is enabled by the output of the preceding flip-flop.

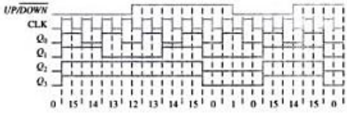
2. A modulus-14 counter has fourteen states requiring four flip-flops.

8–5 See Table 8–13.

8–6 Application of Boolean algebra to the logic in Figure 8–37 shows that the output of each OR gate agrees with the expression in step 5.


CLK A
CLK B
C
C
RO(1)
RO(2)
74LS93A
Q_0 Q_1 Q_2 Q_3

▲ **FIGURE 8–75**


$UP/\overline{DOWN}$
CLK
Q_0
Q_1
Q_2
Q_3
0 15 14 13 12 13 14 15 0 1 0 15 14 15 0

▲ **FIGURE 8–76**

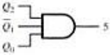
▼ **TABLE 8–13**

PRESENT INVALID STATE			J-K INPUTS						NEXT STATE			
Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2	Q_1	Q_0	
0	0	0	0	0	1	1	1	1	0	1	1	
0	1	1	1	1	1	1	1	1	1	0	0	
1	0	0	0	0	1	1	1	0	1	1	1	valid state
1	1	0	1	1	1	1	1	0	0	0	1	valid state

8–7 Five decade counters are required. $10^5 = 100{,}000$

8–8 $f_{Q0} = 1\text{ MHz}/[(10)(2)] = 50\text{ kHz}$

8–9 See Figure 8–77.


Q_2
$\overline{Q}_1$
Q_0
5

▲ **FIGURE 8–77**

SELF-TEST

1. (a) **2.** (b) **3.** (b) **4.** (c) **5.** (a) **6.** (c) **7.** (b) **8.** (c)

9. (d) **10.** (a) **11.** (c) **12.** (b) **13.** (b) **14.** (d)

9

SHIFT REGISTERS

CHAPTER OBJECTIVES

- Identify the basic forms of data movement in shift registers
- Explain how serial in/serial out, serial in/parallel out, parallel in/serial out, and parallel in/parallel out shift registers operate
- Describe how a bidirectional shift register operates
- Determine the sequence of a Johnson counter
- Set up a ring counter to produce a specified sequence
- Construct a ring counter from a shift register
- Use a shift register as a time-delay device
- Use a shift register to implement a serial-to-parallel data converter
- Implement a basic shift-register-controlled keyboard encoder

INTRODUCTION

Shift registers are a type of sequential logic circuit closely related to digital counters. Registers are used primarily for the storage of digital data and typically do not possess a characteristic internal sequence of states as do counters. There are exceptions, however, and these are covered in Section 9–7.

In this chapter, the basic types of shift registers are studied and several applications are presented.

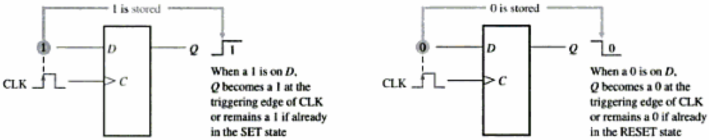
9–1 BASIC SHIFT REGISTER FUNCTIONS

Shift registers consist of an arrangement of flip-flops and are important in applications involving the storage and transfer of data in a digital system. A register, unlike a counter, has no specified sequence of states, except in certain very specialized applications. A register, in general, is used solely for storing and shifting data (1s and 0s) entered into it from an external source and typically possesses no characteristic internal sequence of states.

After completing this section, you should be able to

- Explain how a flip-flop stores a data bit
- Define the storage capacity of a shift register
- Define the shifting capability of a register

A **register** is a digital circuit with two basic functions: data storage and data movement. The storage capability of a register makes it an important type of memory device. A register can consist of one or more flip-flops used to store and shift data. Figure 9–1 illustrates the concept of storing a 1 or a 0 in a D flip-flop. A 1 is applied to the data input as shown, and a clock pulse is applied that stores the 1 by *setting* the flip-flop. When the 1 on the input is removed, the flip-flop remains in the SET state, thereby storing the 1. A similar procedure applies to the storage of a 0 by *resetting* the flip-flop, as also illustrated in Figure 9–1.



When a 1 is on *D*, *Q* becomes a 1 at the triggering edge of CLK or remains a 1 if already in the SET state

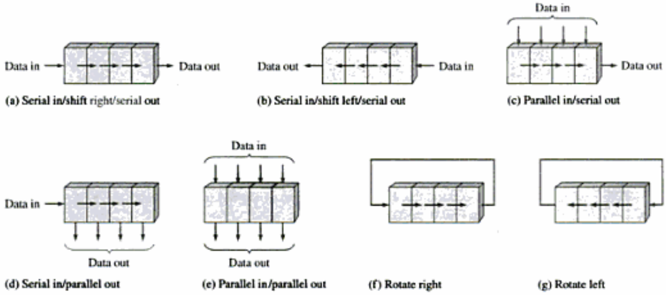
When a 0 is on *D*, *Q* becomes a 0 at the triggering edge of CLK or remains a 0 if already in the RESET state

▲ **FIGURE 9–1**

The flip-flop as a storage element.

The *storage capacity* of a register is the total number of bits (1s and 0s) of digital data it can retain. Each **stage** (flip-flop) in a **shift** register represents one bit of storage capacity; therefore, the number of stages in a register determines its storage capacity.

The shifting capability of a register permits the movement of data from stage to stage within the register or into or out of the register upon application of clock pulses. Figure 9–2 illustrates the types of data movement in shift registers. The block represents any arbitrary 4-bit register, and the arrows indicate the direction of data movement.



(a) Serial in/shift right/serial out

(b) Serial in/shift left/serial out

(c) Parallel in/serial out

(d) Serial in/parallel out

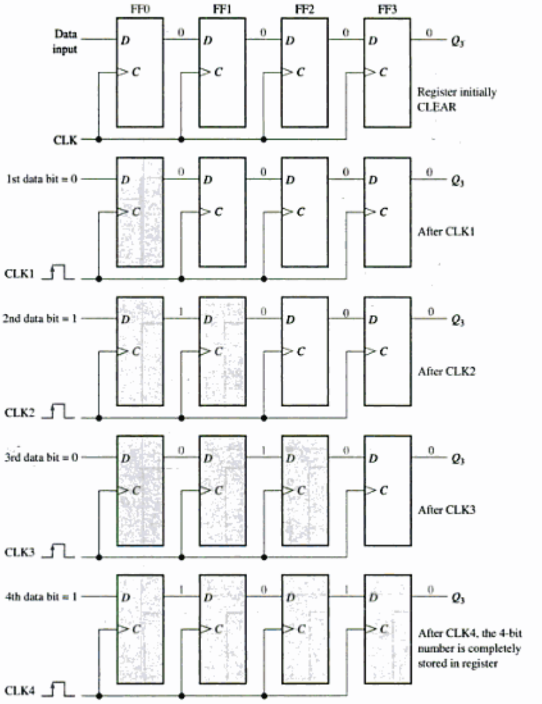
(e) Parallel in/parallel out

(f) Rotate right

(g) Rotate left

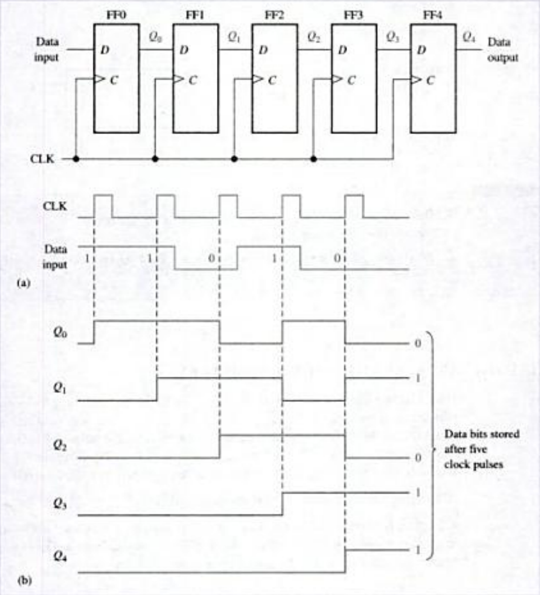
▲ **FIGURE 9–2**

Basic data movement in shift registers.


FF0
FF1
FF2
FF3
Data input
CLK
D
C
0
Q_3
Register initially CLEAR
1st data bit = 0
CLK1
After CLK1
2nd data bit = 1
CLK2
After CLK2
3rd data bit = 0
CLK3
After CLK3
4th data bit = 1
CLK4
After CLK4, the 4-bit number is completely stored in register

▲ **FIGURE 9–4**

If you want to get the data out of the register, the bits must be shifted out serially and taken off the Q_3 output, as Figure 9–5 illustrates. After CLK4 in the data-entry operation just described, the right-most bit, 0, appears on the Q_3 output. When clock pulse CLK5 is applied, the second bit appears on the Q_3 output. Clock pulse CLK6 shifts the third bit to the output, and CLK7 shifts the fourth bit to the output. Notice that while the original four bits are being shifted out, more bits can be shifted in. All zeros are shown being shifted in.


FF0
FF1
FF2
FF3
FF4
Data input
D
C
Q_0
Q_1
Q_2
Q_3
Q_4
Data output
CLK
CLK
Data input
1
1
0
1
0
(a)
Q_0
Q_1
Q_2
Q_3
Q_4
0
1
0
1
1
Data bits stored after five clock pulses
(b)

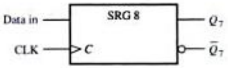
▲ FIGURE 9–6

Solution The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains $Q_4Q_3Q_2Q_1Q_0 = 11010$ after five clock pulses. See Figure 9–6(b).

Supplementary Problem Show the states of the register if the data input is inverted. The register is initially cleared.

A traditional logic block symbol for an 8-bit serial in/serial out shift register is shown in Figure 9–7. The "SRG 8" designation indicates a shift register (SRG) with an 8-bit capacity.

▶ **FIGURE 9–7**



SECTION 9–2 REVIEW

1. Develop the logic diagram for the shift register in Figure 9–3, using J-K flip-flops to replace the D flip-flops.
2. How many clock pulses are required to enter a byte of data serially into an 8-bit shift register?

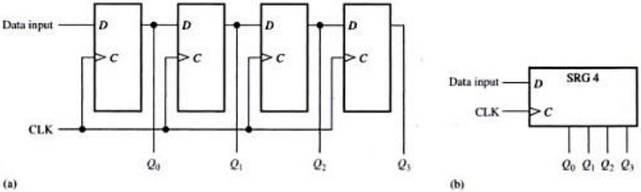
9–3 SERIAL IN/PARALLEL OUT SHIFT REGISTERS

Data bits are entered serially (right-most bit first) into this type of register in the same manner as discussed in Section 9–2. The difference is the way in which the data bits are taken out of the register; in the parallel output register, the output of each stage is available. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output.

After completing this section, you should be able to

■ Explain how data bits are taken out of a shift register in parallel ■ Compare serial output to parallel output ■ Discuss the 74HC164 8-bit shift register ■ Develop and analyze timing diagrams for serial in/parallel out registers

Figure 9–8 shows a 4-bit serial in/parallel out shift register and its logic block symbol.



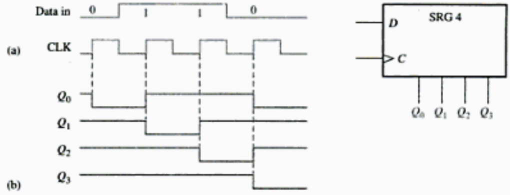
▲ **FIGURE 9–8**

A serial in/parallel out shift register

EXAMPLE 9–2

Show the states of the 4-bit register (SRG 4) for the data input and clock waveforms in Figure 9–9(a). The register initially contains all 1s.

FIGURE 9–9


Data in
0
1
1
0
(a)
CLK
Q_0
Q_1
Q_2
(b)
Q_3
D
SRG 4
C
Q_0 Q_1 Q_2 Q_3

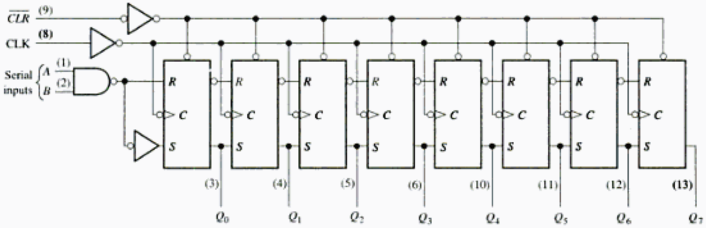
Solution The register contains 0110 after four clock pulses. See Figure 9–9(b).

Supplementary Problem If the data input remains 0 after the fourth clock pulse, what is the state of the register after three additional clock pulses?

AN 8-BIT SERIAL IN/PARALLEL OUT SHIFT REGISTER

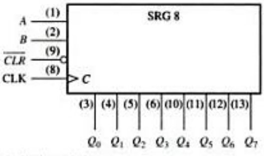
The 74HC164 is an example of an IC shift register having serial in/parallel out operation. The logic diagram is shown in Figure 9–10(a), and a typical logic block symbol is shown in part (b). Notice that this device has two gated serial inputs, *A* and *B*, and a clear ($\overline{CLR}$) input that is active-LOW. The parallel outputs are Q_0 through Q_7.

A sample timing diagram for the 74HC164 is shown in Figure 9–11. Notice that the serial input data on input *A* are shifted into and through the register after input *B* goes HIGH.


$\overline{CLR}$
(9)
CLK
(8)
Serial inputs
A
(1)
B
(2)
R
C
S
(3)
(4)
(5)
(6)
(10)
(11)
(12)
(13)
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7

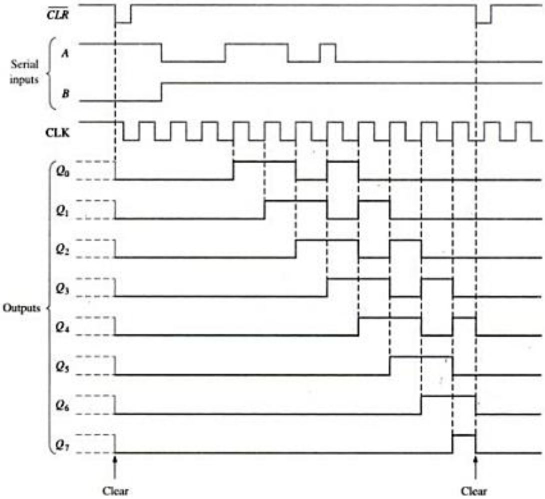
(a) Logic diagram

FIGURE 9–10


SRG 8
A
B
CLR
CLK
(1)
(2)
(9)
(8)
C
(3) (4) (5) (6) (10) (11) (12) (13)
Q_0 Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7

(b) Logic symbol

▲ FIGURE 9–10 (continued)


CLR
Serial inputs
A
B
CLK
Outputs
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7
Clear
Clear

▲ FIGURE 9–11

SECTION 9–3 REVIEW

1. The bit sequence 1101 is serially entered (right-most bit first) into a 4-bit parallel out shift register that is initially clear. What are the Q outputs after two clock pulses?
2. How can a serial in/parallel out register be used as a serial in/serial out register?

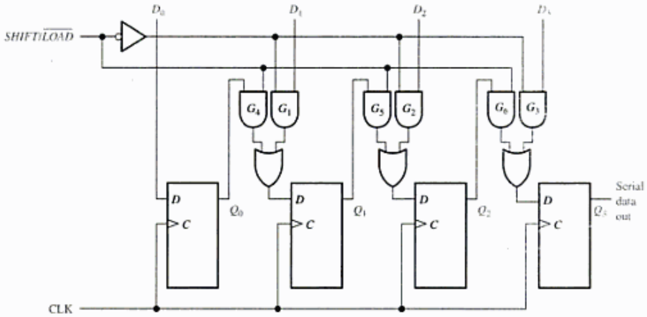
9–4 PARALLEL IN/SERIAL OUT SHIFT REGISTERS

For a register with parallel data inputs, the bits are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis on one line as with serial data inputs. The serial output is the same as described in Section 9–2, once the data are completely stored in the register.

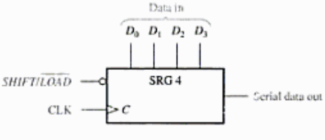
After completing this section, you should be able to

- Explain how data bits are entered into a shift register in parallel
- Compare serial input to parallel input
- Discuss the 74HC165 8-bit parallel-load shift register
- Develop and analyze timing diagrams for parallel in/serial out registers

Figure 9–12 illustrates a 4-bit parallel in/serial out shift register and a typical logic symbol. Notice that there are four data-input lines, D_0, D_1, D_2, and D_3, and a *SHIFT*/$\overline{LOAD}$ input, which allows four bits of data to load in parallel into the register. When *SHIFT*/$\overline{LOAD}$ is LOW, gates G_1 through G_3 are enabled, allowing each data bit to be applied to the D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with $D = 1$ will set and those with $D = 0$ will reset, thereby storing all four bits simultaneously.


D_0
D_1
D_2
D_3
SHIFT/$\overline{LOAD}$
G_4
G_1
G_5
G_2
G_6
G_3
D
C
Q_0
Q_1
Q_2
Q_3
Serial data out
CLK

(a) Logic diagram


Data in
D_0 D_1 D_2 D_3
SHIFT/$\overline{LOAD}$
SRG 4
CLK
C
Serial data out

(b) Logic symbol

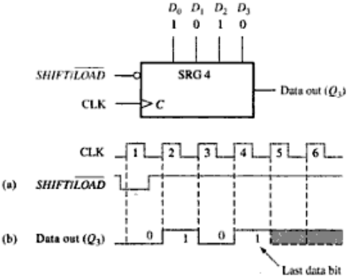
▲ **FIGURE 9–12**

When $SHIFT/\overline{LOAD}$ is HIGH, gates G_1 through G_3 are disabled and gates G_4 through G_6 are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on the $SHIFT/\overline{LOAD}$ input.

EXAMPLE 9–3

Show the data-output waveform for a 4-bit register with the parallel input data and the clock and $SHIFT/\overline{LOAD}$ waveforms given in Figure 9–13(a). Refer to Figure 9–12(a) for the logic diagram.

▶ FIGURE 9–13



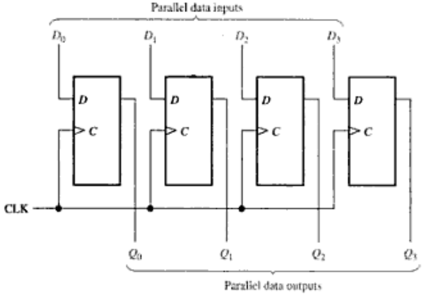
Solution On clock pulse 1, the parallel data ($D_0D_1D_2D_3$ = 1010) are loaded into the register, making Q_3 a 0. On clock pulse 2 the 1 from Q_2 is shifted onto Q_3; on clock pulse 3 the 0 is shifted onto Q_3; on clock pulse 4 the last data bit (1) is shifted onto Q_3; and on clock pulse 5, all data bits have been shifted out, and only 1s remain in the register (assuming the D input remains a 1). See Figure 9–13(b).

Supplementary Problem Show the data-output waveform for the clock and $SHIFT/\overline{LOAD}$ inputs shown in Figure 9–13(a) if the parallel data are $D_0D_1D_2D_3$ = 0101.

AN 8-BIT PARALLEL LOAD SHIFT REGISTER

The 74HC165 is an example of an IC shift register that has a parallel in/serial out operation (it can also be operated as serial in/serial out). Figure 9–14(a) shows the internal logic diagram for this device, and part (b) shows a typical logic block symbol. A LOW on the $SHIFT/\overline{LOAD}$ input ($SH/\overline{LD}$) enables all the NAND gates for parallel loading. When an input data bit is a 1, the flip-flop is asynchronously set by a LOW out of the upper gate. When an input data bit is a 0, the flip-flop is asynchronously reset by a LOW out of the lower gate. Additionally, data can be entered serially on the *SER* input. Also, the clock can be inhibited anytime with a HIGH on the *CLK INH* input. The serial data outputs of the register are Q_7 and its complement $\overline{Q}_7$. This implementation is different from the synchronous method of parallel loading previously discussed, demonstrating that there are usually several ways to accomplish the same function.

Figure 9–16 shows a parallel in/parallel out register.

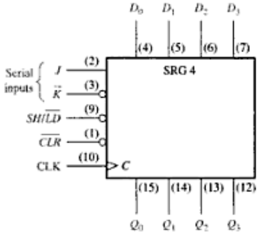

Parallel data inputs
D_0
D_1
D_2
D_3
D
C
CLK
Q_0
Q_1
Q_2
Q_3
Parallel data outputs

▲ **FIGURE 9–16**

A 4-BIT PARALLEL-ACCESS SHIFT REGISTER

The 74HC195 can be used for parallel in/parallel out operation. Because it also has a serial input, it can be used for serial in/serial out and serial in/parallel out operations. It can be used for parallel in/serial out operation by using Q_3 as the output. A typical logic block symbol is shown in Figure 9–17.

▶ **FIGURE 9–17**


D_0
D_1
D_2
D_3
(4)
(5)
(6)
(7)
SRG 4
Serial inputs
J
$\overline{K}$
(2)
(3)
$SH/\overline{LD}$
(9)
$\overline{CLR}$
(1)
CLK
(10)
C
(15)
(14)
(13)
(12)
Q_0
Q_1
Q_2
Q_3

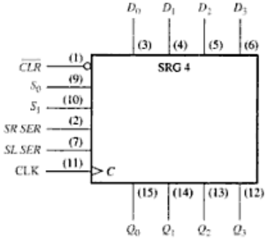
When the $SHIFT/\overline{LOAD}$ input ($SH/\overline{LD}$) is LOW, the data on the parallel inputs are entered synchronously on the positive transition of the clock. When $SH/\overline{LD}$ is HIGH, stored data will shift right (Q_0 to Q_3) synchronously with the clock. Inputs J and $\overline{K}$ are the serial data inputs to the first stage of the register (Q_0); Q_3 can be used for serial output data. The active-LOW clear input is asynchronous.

The timing diagram in Figure 9–18 illustrates the operation of this register.

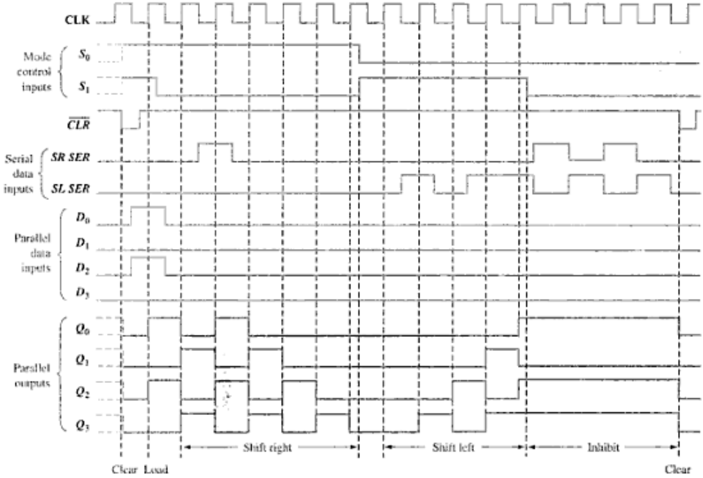
A 4-BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER

The 74HC194 is an example of a universal bidirectional shift register in integrated circuit form. A **universal shift register** has both serial and parallel input and output capability. A logic block symbol is shown in Figure 9–21, and a sample timing diagram is shown in Figure 9–22.

▶ **FIGURE 9–21**



D_0
D_1
D_2
D_3
(3)
(4)
(5)
(6)
SRG 4
$\overline{CLR}$
(1)
S_0
(9)
S_1
(10)
SR SER
(2)
SL SER
(7)
CLK
(11)
C
(15)
(14)
(13)
(12)
Q_0
Q_1
Q_2
Q_3



CLK
Mode control inputs
S_0
S_1
$\overline{CLR}$
Serial data inputs
SR SER
SL SER
Parallel data inputs
D_0
D_1
D_2
D_3
Parallel outputs
Q_0
Q_1
Q_2
Q_3
Shift right
Shift left
Inhibit
Clear
Load
Clear

▲ **FIGURE 9–22**

Parallel loading, which is synchronous with a positive transition of the clock, is accomplished by applying the four bits of data to the parallel inputs and a HIGH to the S_0 and S_1 inputs. Shift right is accomplished synchronously with the positive edge of the clock when S_0 is HIGH and S_1 is LOW. Serial data in this mode are entered at the shift-right serial input (*SR SER*). When S_0 is LOW and S_1 is HIGH, data bits shift left synchronously with the clock, and new data are entered at the shift-left serial input (*SL SER*). Input *SR SER* goes into the Q_0 stage, and *SL SER* goes into the Q_3 stage.

SECTION 9–6 REVIEW

1. Assume that the 4-bit bidirectional shift register in Figure 9–19 has the following contents: $Q_0 = 1$, $Q_1 = 1$, $Q_2 = 0$, and $Q_3 = 0$. There is a 1 on the serial data-input line. If $RIGHT/\overline{LEFT}$ is HIGH for three clock pulses and LOW for two more clock pulses, what are the contents after the fifth clock pulse?

9–7 SHIFT REGISTER COUNTERS

A shift register counter is basically a shift register with the serial output connected back to the serial input to produce special sequences. These devices are often classified as counters because they exhibit a specified sequence of states. Two of the most common types of shift register counters, the Johnson counter and the ring counter, are introduced in this section.

After completing this section, you should be able to

- Discuss how a shift register counter differs from a basic shift register
- Explain the operation of a Johnson counter
- Specify a Johnson sequence for any number of bits
- Explain the operation of a ring counter and determine the sequence of any specific ring counter

The Johnson Counter

In a **Johnson counter** the complement of the output of the last flip-flop is connected back to the *D* input of the first flip-flop (it can be implemented with other types of flip-flops as well). This feedback arrangement produces a characteristic sequence of states, as shown in Table 9–1 for a 4-bit device and in Table 9–2 for a 5-bit device. Notice that the 4-bit sequence has a total of eight states, or bit patterns, and that the 5-bit sequence has a total of ten states. In general, a Johnson counter will produce a modulus of $2n$, where n is the number of stages in the counter.

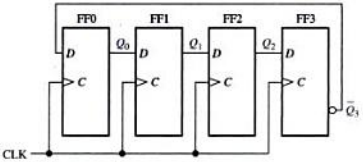
TABLE 9–1

CLOCK PULSE	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

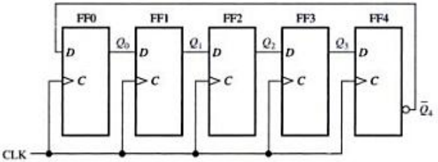
▶ TABLE 9–2

CLOCK PULSE	Q_0	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

The implementations of the 4-stage and 5-stage Johnson counters are shown in Figure 9–23. The implementation of a Johnson counter is very straightforward and is the same regardless of the number of stages. The Q output of each stage is connected to the D input of the next stage (assuming that D flip-flops are used). The single exception is that the $\overline{Q}$ output of the last stage is connected back to the D input of the first stage. As the sequences in Table 9–1 and 9–2 show, the counter will "fill up" with 1s from left to right, and then it will "fill up" with 0s again.



(a) Four-bit Johnson counter

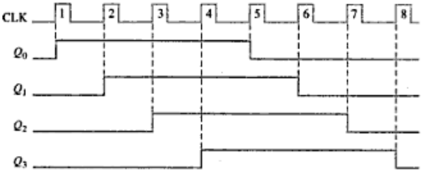


(b) Five-bit Johnson counter

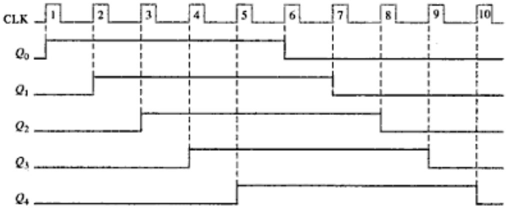
▲ FIGURE 9–23

Diagrams of the timing operations of the 4-bit and 5-bit counters are shown in Figures 9–24 and 9–25, respectively.

▶ FIGURE 9–24



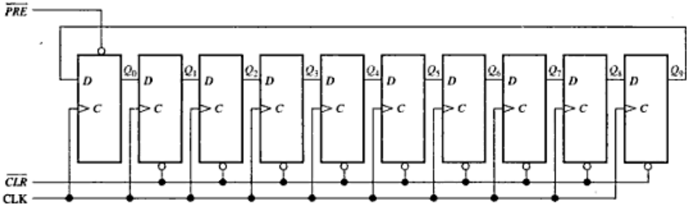
▶ FIGURE 9–25



The Ring Counter

The **ring counter** utilizes one flip-flop for each state in its sequence. It has the advantage that decoding gates are not required. In the case of a 10-bit ring counter, there is a unique output for each decimal digit.

A logic diagram for a 10-bit ring counter is shown in Figure 9–26. The sequence for this ring counter is given in Table 9–3. Initially, a 1 is preset into the first flip-flop, and the rest of the flip-flops are cleared. Notice that the interstage connections are the same as those for a Johnson counter, except that Q rather than $\overline{Q}$ is fed back from the last stage. The ten outputs of the counter indicate directly the decimal count of the clock pulse. For instance, a 1 on Q_0 represents a zero, a 1 on Q_1 represents a one, a 1 on Q_2 represents a two, a 1 on Q_3 represents a three, and so on. You should verify for yourself that the 1 is always retained in the counter and simply shifted "around the ring," advancing one stage for each clock pulse.



▲ FIGURE 9–26

TABLE 9-3

CLOCK PULSE	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Modified sequences can be achieved by having more than a single 1 in the counter, as illustrated in Example 9–5.

EXAMPLE 9–5

If a 10-bit ring counter similar to Figure 9–26 has the initial state 1010000000, determine the waveform for each of the Q outputs.

Solution See Figure 9–27.

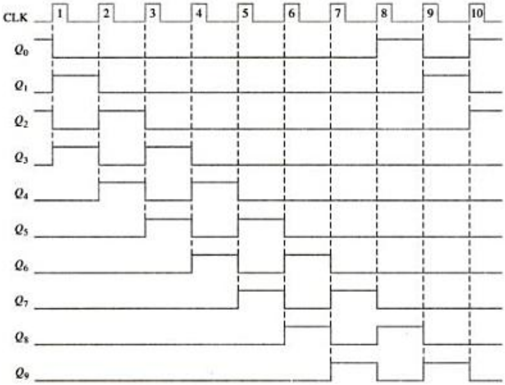


FIGURE 9–27

Supplementary Problem If a 10-bit ring counter has an initial state 0101001111, determine the waveform for each Q output.

SECTION 9–7 REVIEW

1. How many states are there in an 8-bit Johnson counter sequence?
2. Write the sequence of states for a 3-bit Johnson counter starting with 000.

9–8 SHIFT REGISTER APPLICATIONS

Shift registers are found in many types of applications, a few of which are presented in this section.

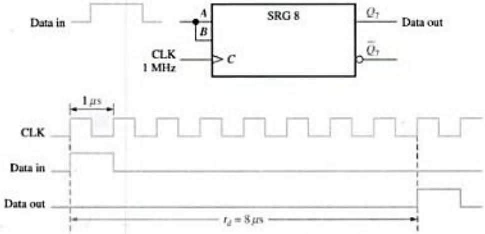
After completing this section, you should be able to

- Use a shift register to generate a time delay
- Implement a specified ring counter sequence using a 74HC195 shift register
- Discuss how shift registers are used for serial-to-parallel conversion of data
- Define *UART*
- Explain the operation of a keyboard encoder and how registers are used in this application

Time Delay

The serial in/serial out shift register can be used to provide a time delay from input to output, that is, a function of both the number of stages (n) in the register and the clock frequency.

When a data pulse is applied to the serial input as shown in Figure 9–28 (A and B connected together), it enters the first stage on the triggering edge of the clock pulse. It is then shifted from stage to stage on each successive clock pulse until it appears on the serial output n clock periods later. This time-delay operation is illustrated in Figure 9–28, in which an 8-bit serial in/serial out shift register is used with a clock frequency of 1 MHz to achieve a time delay (t_d) of 8 μs (8 × 1 μs). This time can be adjusted up or down by changing the clock frequency. The time delay can also be increased by cascading shift registers and decreased by taking the outputs from successively lower stages in the register if the outputs are available, as illustrated in Example 9–6.

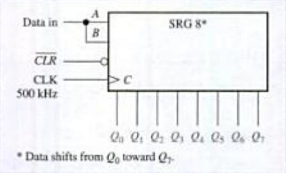


▲ FIGURE 9–28

EXAMPLE 9–6

Determine the amount of time delay between the serial input and each output in Figure 9–29. Show a timing diagram to illustrate.

FIGURE 9–29


Data in
A
B
SRG 8*
$\overline{CLR}$
CLK
500 kHz
C
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7

* Data shifts from Q_0 toward Q_7.

Solution The clock period is 2 μs. Thus, the time delay can be increased or decreased in 2 μs increments from a minimum of 2 μs to a maximum of 16 μs, as illustrated in Figure 9–30.

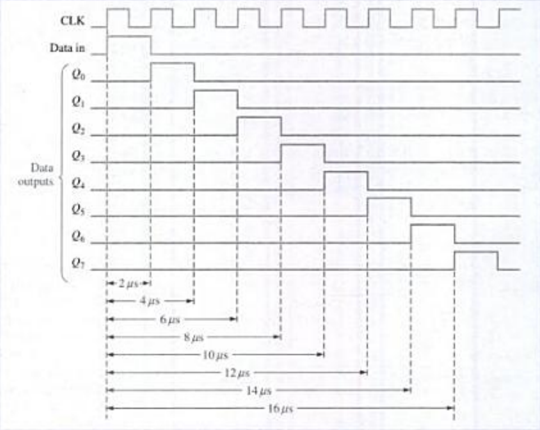

CLK
Data in
Data outputs
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7
2 μs
4 μs
6 μs
8 μs
10 μs
12 μs
14 μs
16 μs

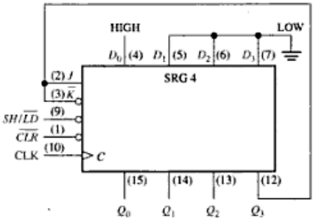
FIGURE 9–30

Supplementary Problem Determine the clock frequency required to obtain a time delay of 24 μs to the Q_7 output in Figure 9–29.

A RING COUNTER USING A 74HC195 SHIFT REGISTER

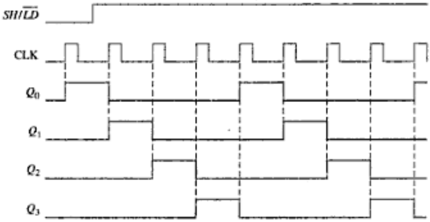
If the output is connected back to the serial input, a shift register can be used as a ring counter. Figure 9–31 illustrates this application with a 74HC195 4-bit shift register.

FIGURE 9–31



Initially, a bit pattern of 1000 (or any other pattern) can be synchronously preset into the counter by applying the bit pattern to the parallel data inputs, taking the $SH/\overline{LD}$ input LOW, and applying a clock pulse. After this initialization, the 1 continues to circulate through the ring counter, as the timing diagram in Figure 9–32 shows.

FIGURE 9–32



Serial-to-Parallel Data Converter

Serial data transmission from one digital system to another is commonly used to reduce the number of wires in the transmission line. For example, eight bits can be sent serially over one wire, but it takes eight wires to send the same data in parallel.

A computer or microprocessor-based system commonly requires incoming data to be in parallel format, thus the requirement for serial-to-parallel conversion. A simplified serial-to-parallel data converter, in which two types of shift registers are used, is shown in Figure 9–33.

To illustrate the operation of this serial-to-parallel converter, the serial data format shown in Figure 9–34 is used. It consists of eleven bits. The first bit (start bit) is always 0 and always begins with a HIGH-to-LOW transition. The next eight bits (D_7 through D_0) are the data bits (one of the bits can be parity), and the last two bits (stop bits) are always 1s. When no data are being sent, there is a continuous 1 on the serial data line.

FIGURE 9–35

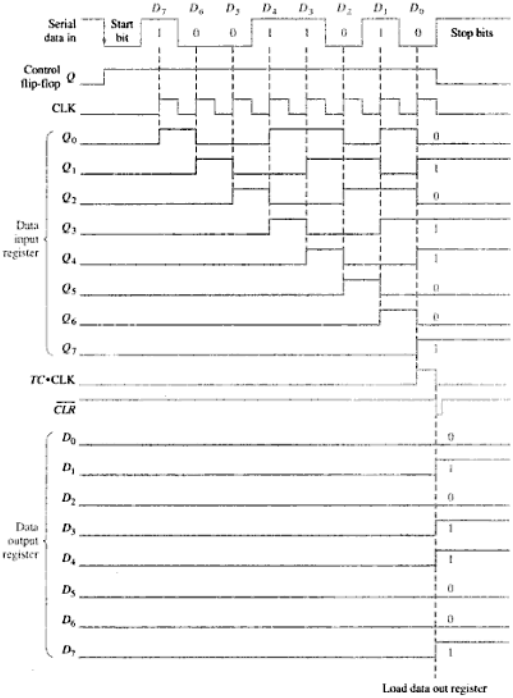
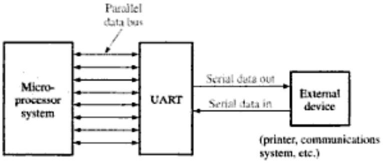

D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
Serial data in
Start bit
1
0
0
1
1
0
1
0
Stop bits
Control flip-flop Q
CLK
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7
Data input register
0
1
0
1
1
0
0
1
TC•CLK
$\overline{CLR}$
D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
Data output register
0
1
0
1
1
0
0
1
Load data out register

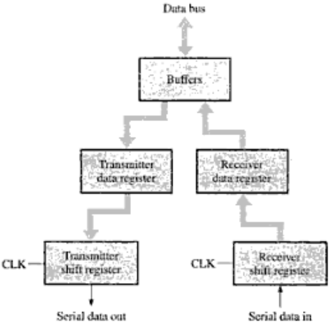
FIGURE 9–36


Parallel data bus
Micro-processor system
UART
Serial data out
Serial data in
External device
(printer, communications system, etc.)

A UART includes a serial-to-parallel data converter such as we have discussed and a parallel-to-serial converter, as shown in Figure 9–37. The data bus is basically a set of parallel conductors along which data move between the UART and the microprocessor system. Buffers interface the data registers with the data bus.

▶ **FIGURE 9–37**

Basic UART block diagram.



The UART receives data in serial format, converts the data to parallel format, and places them on the data bus. The UART also accepts parallel data from the data bus, converts the data to serial format, and transmits them to an external device.

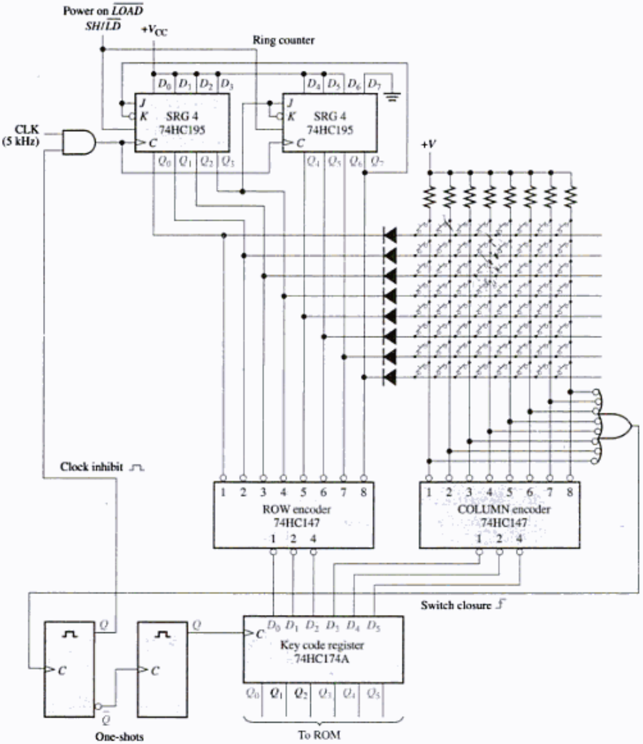
Keyboard Encoder

The keyboard encoder is a good example of the application of a shift register used as a ring counter in conjunction with other devices. Recall that a simplified computer keyboard encoder without data storage was presented in Chapter 6.

Figure 9–38 shows a simplified keyboard encoder for encoding a key closure in a 64-key matrix organized in eight rows and eight columns. Two 74HC195 4-bit shift registers are connected as an 8-bit ring counter with a fixed bit pattern of seven 1s and one 0 preset into it when the power is turned on. Two 74HC147 priority encoders (introduced in Chapter 6) are used as eight-line-to-three-line encoders (9 input HIGH, 8 output unused) to encode the ROW and COLUMN lines of the keyboard matrix. The 74HC174A (hex flip-flops) is used as a parallel in/parallel out register in which the ROW/COLUMN code from the priority encoders is stored.

The basic operation of the keyboard encoder in Figure 9–38 is as follows: The ring counter "scans" the rows for a key closure as the clock signal shifts the 0 around the counter at a 5 kHz rate. The 0 (LOW) is sequentially applied to each ROW line, while all other ROW lines are HIGH. All the ROW lines are connected to the ROW encoder inputs, so the 3-bit output of the ROW encoder at any time is the binary representation of the ROW line that is LOW. When there is a key closure, one COLUMN line is connected to one ROW line. When the ROW line is taken LOW by the ring counter, that particular COLUMN line is also pulled LOW. The COLUMN encoder produces a binary output corresponding to the COLUMN in which the key is closed. The 3-bit ROW code plus the 3-bit COLUMN code uniquely identifies the key that is closed. This 6-bit code is applied to the inputs of the key code register. When a key is closed, the two one-shots produce a delayed clock pulse to parallel-load the 6-bit code into the key code register. This delay allows the contact bounce to die out. Also, the first one-shot output inhibits the ring counter to prevent it from scanning while the data are being loaded into the key code register.

The 6-bit code in the key code register is now applied to a ROM (read-only memory) to be converted to an appropriate alphanumeric code that identifies the keyboard character. ROMs are studied in Chapter 10.


Power on $\overline{LOAD}$
$SH/\overline{LD}$
$+V_{CC}$
Ring counter
D_0 D_1 D_2 D_3
D_4 D_5 D_6 D_7
J
K
SRG 4
74HC195
C
CLK
(5 kHz)
Q_0 Q_1 Q_2 Q_3
Q_4 Q_5 Q_6 Q_7
+V
Clock inhibit
1 2 3 4 5 6 7 8
ROW encoder
74HC147
1 2 4
1 2 3 4 5 6 7 8
COLUMN encoder
74HC147
1 2 4
Switch closure
Q
Q
$\bar{Q}$
C
C
D_0 D_1 D_2 D_3 D_4 D_5
Key code register
74HC174A
Q_0 Q_1 Q_2 Q_3 Q_4 Q_5
One-shots
To ROM

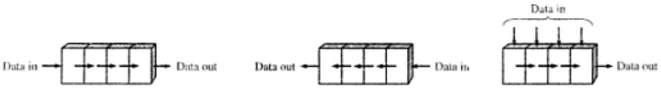
▲ **FIGURE 9–38**

SECTION 9–8 REVIEW

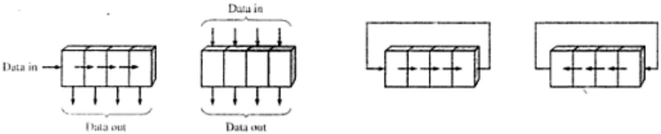
1. In the keyboard encoder, how many times per second does the ring counter scan the keyboard?
2. What is the 6-bit ROW/COLUMN code (key code) for the top row and the left-most column in the keyboard encoder?
3. What is the purpose of the diodes in the keyboard encoder? What is the purpose of the resistors?

SUMMARY

- The basic types of data movement in shift registers are illustrated in Figure 9–39.



(a) Serial in/shift right/serial out (b) Serial in/shift left/serial out (c) Parallel in/serial out



(d) Serial in/parallel out (e) Parallel in/parallel out (f) Rotate right (g) Rotate left

▲ **FIGURE 9–39**

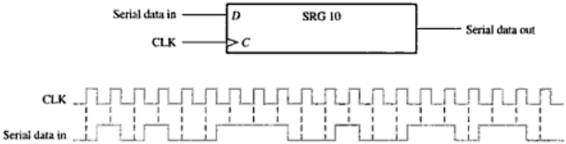
- Shift register counters are shift registers with feedback that exhibit special sequences. Examples are the Johnson counter and the ring counter.
- The Johnson counter has $2n$ states in its sequence, where n is the number of stages.
- The ring counter has n states in its sequence.

SELF-TEST

Answers are at the end of the chapter.

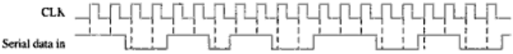
1. A stage in a shift register consists of
   (a) a latch **(b)** a flip-flop **(c)** a byte of storage **(d)** four bits of storage
2. To serially shift a byte of data into a shift register, there must be
   (a) one clock pulse **(b)** one load pulse
   (c) eight clock pulses **(d)** one clock pulse for each 1 in the data
3. To parallel load a byte of data into a shift register with a synchronous load, there must be
   (a) one clock pulse **(b)** one clock pulse for each 1 in the data
   (c) eight clock pulses **(d)** one clock pulse for each 0 in the data
4. The group of bits 10110101 is serially shifted (right-most bit first) into an 8-bit parallel output shift register with an initial state of 11100100. After two clock pulses, the register contains
   (a) 01011110 **(b)** 10110101 **(c)** 01111001 **(d)** 00101101
5. With a 100 kHz clock frequency, eight bits can be serially entered into a shift register in
   (a) 80 μs **(b)** 8 μs **(c)** 80 ms **(d)** 10 μs
6. With a 1 MHz clock frequency, eight bits can be parallel entered into a shift register
   (a) in 8 μs **(b)** in the propagation delay time of eight flip-flops
   (c) in 1 μs **(d)** in the propagation delay time of one flip-flop

6. For the serial in/serial out shift register, determine the data-output waveform for the data-input and clock waveforms in Figure 9–43. Assume that the register is initially cleared.


Serial data in
CLK
D
C
SRG 10
Serial data out
CLK
Serial data in

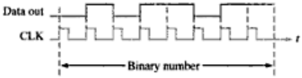
▲ FIGURE 9–43

7. Solve Problem 6 for the waveforms in Figure 9–44.


CLK
Serial data in

▶ FIGURE 9–44

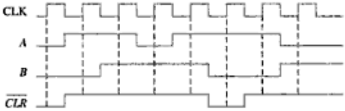
8. A leading-edge clocked serial in/serial out shift register has a data-output waveform as shown in Figure 9–45. What binary number is stored in the 8-bit register if the first data bit out (left-most) is the LSB?


Data out
CLK
t
Binary number

▶ FIGURE 9–45

SECTION 9–3 Serial In/Parallel Out Shift Registers

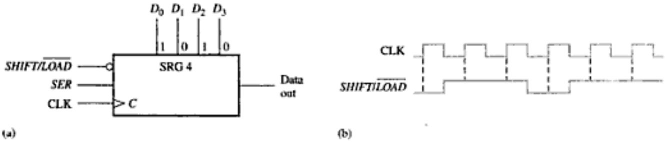
9. Show a complete timing diagram showing the parallel outputs for the shift register in Figure 9–8. Use the waveforms in Figure 9–43 with the register initially clear.
10. Solve Problem 9 for the input waveforms in Figure 9–44.
11. Develop the Q_0 through Q_7 outputs for a 74HC164 shift register with the input waveforms shown in Figure 9–46.


CLK
A
B
$\overline{CLR}$

▶ FIGURE 9–46

SECTION 9–4 Parallel In/Serial Out Shift Registers

12. The shift register in Figure 9–47(a) has $SHIFT/\overline{LOAD}$ and CLK inputs as shown in part (b). The serial data input (SER) is a 0. The parallel data inputs are $D_0 = 1$, $D_1 = 0$, $D_2 = 1$, and $D_3 = 0$ as shown. Develop the data-output waveform in relation to the inputs.


D_0 D_1 D_2 D_3
1
0
1
0
SRG 4
$SHIFT/\overline{LOAD}$
SER
CLK
C
Data out
CLK
$SHIFT/\overline{LOAD}$

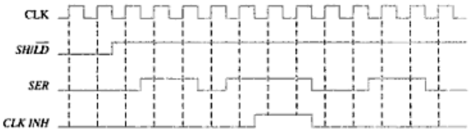
(a)

(b)

▲ **FIGURE 9–47**

13. The waveforms in Figure 9–48 are applied to a 74HC165 shift register. The parallel inputs are all 0. Determine the Q_7 waveform.

▶ **FIGURE 9–48**


CLK
$SH/\overline{LD}$
SER
$CLK\ INH$

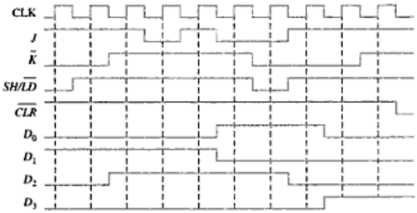
14. Solve Problem 13 if the parallel inputs are all 1.

15. Solve Problem 13 if the SER input is inverted.

SECTION 9–5 Parallel In/Parallel Out Shift Registers

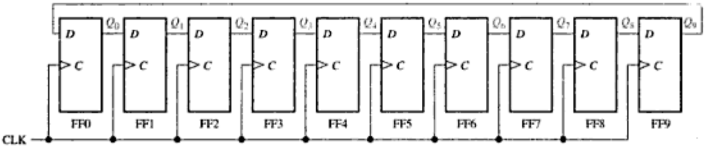
16. Determine all the Q output waveforms for a 74HC195 4-bit shift register when the inputs are as shown in Figure 9–49.

▶ **FIGURE 9–49**


CLK
J
$\overline{K}$
$SH/\overline{LD}$
$\overline{CLR}$
D_0
D_1
D_2
D_3

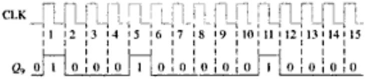
17. Solve Problem 16 if the $SH/\overline{LD}$ input is inverted and the register is initially clear.

18. Use two 74HC195 shift registers to form an 8-bit shift register. Show the required connections.



▲ **FIGURE 9–53**

▶ **FIGURE 9–54**



SECTION 9–8 Shift Register Applications

27. Use 74HC195 4-bit shift registers to implement a 16-bit ring counter. Show the connections.

28. What is the purpose of the power-on $\overline{LOAD}$ input in Figure 9–38?

29. What happens when two keys are pressed simultaneously in Figure 9–38?

ANSWERS

SECTION REVIEWS

SECTION 9–1 Basic Shift Register Functions

1. A counter has a specified sequence of states, but a shift register does not.
2. Storage and data movement are two functions of a shift register.

SECTION 9–2 Serial In/Serial Out Shift Registers

1. FF0: data input to J_0, $\overline{\text{data input}}$ to K_0; FF1: Q_0 to J_1, $\overline{Q}_0$ to K_1; FF2: Q_1 to J_2, $\overline{Q}_1$ to K_2; FF3: Q_2 to J_3, $\overline{Q}_2$ to K_3
2. Eight clock pulses

SECTION 9–3 Serial In/Parallel Out Shift Registers

1. 0100 after 2 clock pulses
2. Take the serial output from the right-most flip-flop for serial out operation.

SECTION 9–4 Parallel In/Serial Out Shift Registers

1. When $SHIFT/\overline{LOAD}$ is HIGH, the data are shifted right one bit per clock pulse. When $SHIFT/\overline{LOAD}$ is LOW, the data on the parallel inputs are loaded into the register.
2. The parallel load operation is asynchronous, so it is not dependent on the clock.

SECTION 9–5 Parallel In/Parallel Out Shift Registers

1. The data outputs are 1001. 2. $Q_0 = 1$ after one clock pulse

SECTION 9–6 **Bidirectional Shift Registers**

1. 1111 after the fifth clock pulse

SECTION 9–7 **Shift Register Counters**

1. Sixteen states are in an 8-bit Johnson counter sequence.
2. For a 3-bit Johnson counter: 000, 100, 110, 111, 011, 001, 000

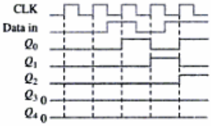
SECTION 9–8 **Shift Register Applications**

1. 625 scans/second **2.** $Q_5Q_4Q_3Q_2Q_1Q_0 = 011011$
3. The diodes provide unidirectional paths for pulling the ROWs LOW and preventing HIGHs on the ROW lines from being connected to the switch matrix. The resistors pull the COLUMN lines HIGH.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

9–1 See Figure 9–55.

FIGURE 9–55

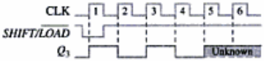

CLK
Data in
Q_0
Q_1
Q_2
Q_3 0
Q_4 0

The output is $Q_4Q_3Q_2Q_1Q_0 =$ 00101 after 5 clock pulses.

9–2 The state of the register after three additional clock pulses is 0000.

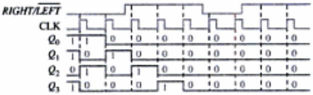
9–3 See Figure 9–56.

FIGURE 9–56


CLK
1
2
3
4
5
6
SHIFT/$\overline{LOAD}$
Q_3
Unknown

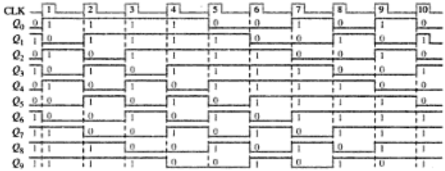
9–4 See Figure 9–57.

FIGURE 9–57


RIGHT/$\overline{LEFT}$
CLK
Q_0
Q_1
Q_2
Q_3
0
1

9–5 See Figure 9–58.

FIGURE 9–58


CLK
1
2
3
4
5
6
7
8
9
10
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7
Q_8
Q_9

9–6 $f = 1/3\ \mu s = 333$ kHz

SELF-TEST

1. (b) **2.** (c) **3.** (a) **4.** (c) **5.** (a) **6.** (d) **7.** (c) **8.** (a)
9. (b) **10.** (c)

10

MEMORY AND STORAGE

CHAPTER OBJECTIVES

- Define the basic memory characteristics
- Explain what a RAM is and how it works
- Explain the difference between static RAMs (SRAMs) and dynamic RAMs (DRAMs)
- Explain what a ROM is and how it works
- Explain how a ROM is programmed
- Describe the various types of PROMs
- Discuss the characteristics of a flash memory
- Describe the expansion of ROMs and RAMs to increase word length and word capacity
- Describe the basic organization of magnetic disks and magnetic tapes
- Describe the basic operation of magneto-optical disks and optical disks
- Describe basic methods for memory testing
- Develop flowcharts for memory testing

INTRODUCTION

Chapter 9 covered shift registers, which are a type of storage device; in fact, a shift register is essentially a small-scale memory. The memory devices covered in this chapter are generally used for longer-term storage of larger amounts of data than registers can provide.

Computers and other types of systems require the permanent or semipermanent storage of large amounts of binary data. Microprocessor-based systems rely on storage devices and memories for their operation because of the necessity for storing programs and for retaining data during processing.

In computer terminology, *memory* usually refers to RAM and ROM and *storage* refers to hard disk, floppy disk, and CD-ROM. In this chapter semiconductor, magnetic, and optical memories are covered.

10–1 BASICS OF SEMICONDUCTOR MEMORY

Memory is the portion of a system for storing binary data in large quantities. Semiconductor memories consist of arrays of storage elements that are generally either latches or capacitors.

After completing this chapter, you should be able to

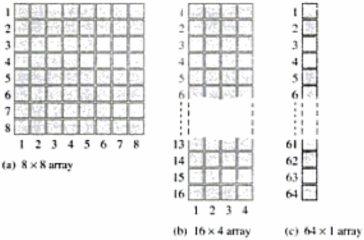
- Explain how a memory stores binary data
- Discuss the basic organization of a memory
- Describe the write operation
- Describe the read operation
- Describe the addressing operation
- Explain what RAMs and ROMs are

Units of Binary Data: Bits, Bytes, Nibbles, and Words

As a rule, memories store data in units that have from one to eight bits. The smallest unit of binary data, as you know, is the **bit**. In many applications, data are handled in an 8-bit unit called a **byte** or in multiples of 8-bit units. The byte can be split into two 4-bit units that are called **nibbles**. A complete unit of information is called a **word** and generally consists of one or more bytes. Some memories store data in 9-bit groups; a 9-bit group consists of a byte plus a parity bit.

The Basic Semiconductor Memory Array

Each storage element in a memory can retain either a 1 or a 0 and is called a **cell**. Memories are made up of arrays of cells, as illustrated in Figure 10–1 using 64 cells as an example. Each block in the **memory array** represents one storage cell, and its location can be identified by specifying a row and a column.



(a) 8 × 8 array (b) 16 × 4 array (c) 64 × 1 array

FIGURE 10–1

A 64-cell memory array organized in three different ways

The 64-cell array can be organized in several ways based on units of data. Figure 10–1(a) shows an 8 × 8 array, which can be viewed as either a 64-bit memory or an 8-byte memory. Part (b) shows a 16 × 4 array, which is a 16-nibble memory, and part (c) shows a 64 × 1 array which is a 64-bit memory. A memory is identified by the number of words it can store times the word size. For example, a 16k × 8 memory can store 16,384 words of eight bits each. The inconsistency here is common in memory terminology. The actual number of words is always a power of 2, which, in this case, is $2^{14} = 16{,}384$. However, it is common practice to state the number to the nearest thousand, in this case, 16k.

Memory Address and Capacity

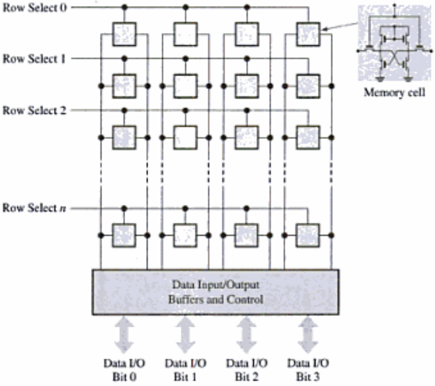
The *location* of a unit of data in a memory array is called its **address**. For example, in Figure 10–2(a), the address of a bit in the array is specified by the row and column as shown. In Figure 10–2(b), the address of a byte is specified only by the row. So, as you can see, the address depends on how the memory is organized into units of data. Personal computers have random-access memories organized in bytes. This means that the smallest group of bits that can be addressed is eight.

Data and $\overline{\text{Data}}$ lines. A data bit is read by taking it off the Data and $\overline{\text{Data}}$ lines. Input data and output data can share the same lines because write and read operations occur at different times.

Basic Static Memory Cell Array The storage cells in a SRAM are organized in rows and columns, as illustrated in Figure 10–8 for the case of an $n \times 4$ array. All the cells in a row share the same Row Select line. Each set of Data and $\overline{\text{Data}}$ lines go to each cell in a given column and are connected to a single data line that serves as both an input and output (Data I/O) through the data input and data output buffers.

▶ FIGURE 10–8

Basic SRAM array



To write a data unit, in this case a nibble, into a given row of cells in the memory array, the Row Select line is taken to its active state and four data bits are placed on the Data Input lines. The Write line is then taken to its active state, which causes each data bit to be stored in a selected cell in the associated column. To read a data unit, the Read line is taken to its active state, which causes the four data bits stored in the selected row to appear on the Data Output lines.

Basic Asynchronous SRAM Organization

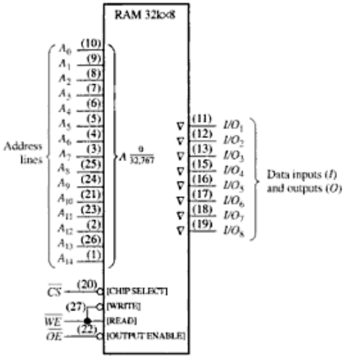
An asynchronous SRAM is one in which the operation is not synchronized with a system clock. To illustrate the general organization of a SRAM, a 32k × 8 bit memory is used. A logic symbol for this memory is shown in Figure 10–9.

In the READ mode, the eight data bits that are stored in a selected address appear on the data output lines. In the WRITE mode, the eight data bits that are applied to the data input lines are stored at a selected address. The data input and data output lines (I/O_1 through I/O_8) are the same lines. During READ, they act as output lines (I_1 through I_8) and during WRITE they act as input lines (O_1 through O_8).

Tristate Outputs and Buses Tristate buffers in a memory allow the data lines to act as either input or output lines and connect the memory to the data bus in a computer. These buffers have three output states: HIGH (1), LOW (0), and HIGH-Z (open). Tristate outputs are indicated on logic symbols by a small inverted triangle (∇), as shown in Figure 10–9, and are used for compatibility with bus structures such as those found in microprocessor-based systems.

FIGURE 10–9

Logic diagram for an asynchronous 32k × 8 SRAM



Physically, a **bus** is a set of conductive paths that serve to interconnect two or more functional components of a system or several diverse systems. Electrically, a bus is a collection of specified voltage levels and/or current levels and signals that allow the various devices connected to the bus to communicate and work properly together.

For example, a microprocessor is connected to memories and input/output devices by certain bus structures. An address bus allows the microprocessor to address the memories, and the data bus provides for transfer of data between the microprocessor, the memories, and the input/output devices such as monitors, printers, keyboards, and modems. The control bus allows the microprocessor to control data transfers and timing for the various components.

Memory Array SRAM chips can be organized in single bits, nibbles (4 bits), bytes (8 bits), or multiple bytes (16, 24, 32 bits, etc.).

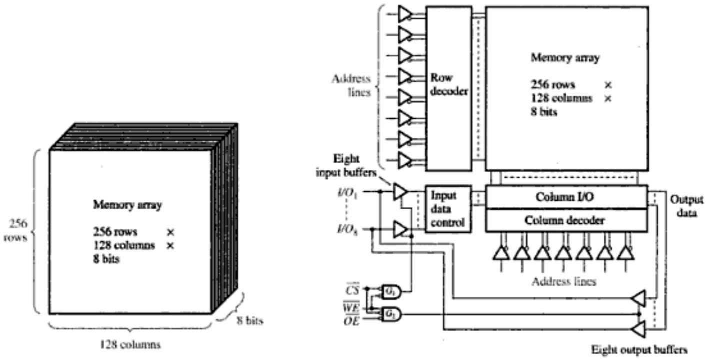
Figure 10–10 shows the organization of a typical 32k × 8 SRAM. The memory cell array is arranged in 256 rows and 128 columns, each with 8 bits, as shown in part (a). There are actually $2^{15} = 32{,}768$ addresses and each address contains 8 bits. The capacity of this example memory is 32,768 bytes (typically expressed as 32 kbytes).

The SRAM in Figure 10–10(b) works as follows. First, the chip select, $\overline{CS}$, must be LOW for the memory to operate. Eight of the fifteen address lines are decoded by the row decoder to select one of the 256 rows. Seven of the fifteen address lines are decoded by the column decoder to select one of the 128 8-bit columns.

READ In the READ mode, the write enable input, $\overline{WE}$, is HIGH and the output enable, $\overline{OE}$, is LOW. The input tristate buffers are disabled by gate G_1, and the column output tristate buffers are enabled by gate G_2. Therefore, the eight data bits from the selected address are routed through the column I/O to the data lines (I/O_1 though I/O_8), which are acting as data output lines.

WRITE In the WRITE mode, $\overline{WE}$ is LOW and $\overline{OE}$ is HIGH. The input buffers are enabled by gate G_1, and the output buffers are disabled by gate G_2. Therefore, the eight input data bits on the data lines are routed through the input data control and the column I/O to the selected address and stored.

Read and Write Cycles Figure 10–11 shows typical timing diagrams for a memory read cycle and a write cycle. For the read cycle shown in part (a), a valid address code is applied to


Memory array
256 rows ×
128 columns ×
8 bits
256 rows
128 columns
8 bits
Address lines
Row decoder
Memory array
256 rows ×
128 columns ×
8 bits
Eight input buffers
I/O1
I/O8
Input data control
Column I/O
Column decoder
Output data
Address lines
CS
WE
OE
G1
G2
Eight output buffers

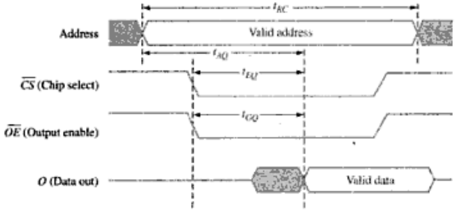
(a) Memory array configuration

(b) Memory block diagram

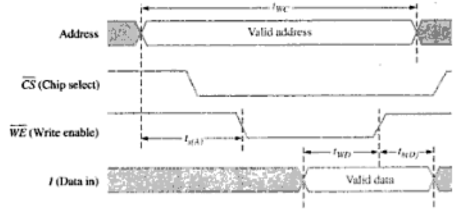
▲ **FIGURE 10–10**

Basic organization of an asynchronous 32k × 8 SRAM

▶ **FIGURE 10–11**


t_{RC}
Address
Valid address
t_{AQ}
t_{EQ}
CS (Chip select)
t_{GQ}
OE (Output enable)
O (Data out)
Valid data

(a) Read cycle ($\overline{WE}$ HIGH)


t_{WC}
Address
Valid address
CS (Chip select)
WE (Write enable)
$t_{s(A)}$
t_{WD}
$t_{h(D)}$
I (Data in)
Valid data

(b) Write cycle ($\overline{WE}$ LOW)

the address lines for a specified time interval called the *read cycle time*, t_{RC}. Next, the chip select ($\overline{CS}$) and the output enable ($\overline{OE}$) inputs go LOW. One time interval after the $\overline{OE}$ input goes LOW, a valid data byte from the selected address appears on the data lines. This time interval is called the *output enable access time*, t_{GQ}.

Two other access times for the read cycle are the *address access time*, t_{AQ}, measured from the beginning of a valid address to the appearance of valid data on the data lines and the *chip enable access time*, t_{EQ}, measured from the HIGH-to-LOW transition of $\overline{CS}$ to the appearance of valid data on the data lines.

During each read cycle, one unit of data, a byte in this case, is read from the memory.

For the write cycle shown in Figure 10–11(b), a valid address code is applied to the address lines for a specified time interval called the *write cycle time*, t_{WC}. Next, the chip select ($\overline{CS}$) and the write enable ($\overline{WE}$) inputs go LOW. The required time interval from the beginning of a valid address until the $\overline{WE}$ input goes LOW is called the *address setup time*, $t_{s(A)}$. The time that the $\overline{WE}$ input must be LOW is the write pulse width. The time that the input $\overline{WE}$ must remain LOW after valid data are applied to the data inputs is designated t_{WD}; the time that the valid input data must remain on the data lines after the $\overline{WE}$ input goes HIGH is the *data hold time*, $t_{h(D)}$.

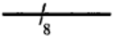
During each write cycle, one unit of data is written into the memory.

Basic Synchronous Burst SRAM Organization

Unlike the asynchronous SRAM, a synchronous SRAM is synchronized with the system clock. For example, in a computer system, the synchronous SRAM operates with the same clock signal that operates the microprocessor so that the microprocessor and memory are synchronized for faster operation.

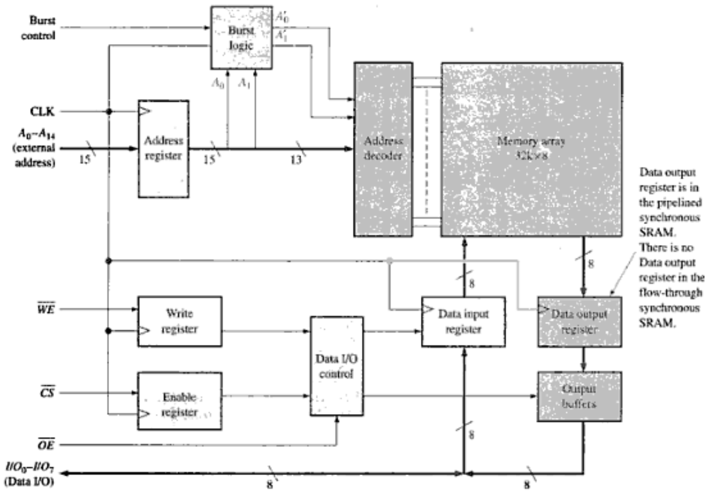
The fundamental concept of the synchronous feature of a SRAM can be shown with Figure 10–12, which is a simplified block diagram of a 32k × 8 memory for purposes of illustration. The synchronous SRAM is very similar to the asynchronous SRAM in terms of the memory array, address decoder, and read/write and enable inputs. The basic difference is that the synchronous SRAM uses clocked registers to synchronize all inputs with the system clock. The address, the read/write input, the chip enable, and the input data are all latched into their respective registers on an active clock pulse edge. Once this information is latched, the memory operation is in sync with the clock.

For the purpose of simplification, a notation for multiple parallel lines or bus lines is introduced in Figure 10–12, as an alternative to drawing each line separately. A set of parallel lines can be indicated by a single heavy line with a slash and the number of separate lines in the set. For example, the following notation represents a set of 8 parallel lines:



The address bits A_0 through A_{14} are latched into the Address register on the positive edge of a clock pulse. On the same clock pulse, the state of the write enable ($\overline{WE}$) line and chip select ($\overline{CS}$) are latched into the Write register and the Enable register respectively. These are one-bit registers or simply flip-flops. Also, on the same clock pulse the input data are latched into the Data input register for a Write operation, and data in a selected memory address are latched into the Data output register for a Read operation, as determined by the Data I/O control based on inputs from the Write register, Enable register, and the Output enable ($\overline{OE}$).

Two basic types of synchronous SRAM are the *flow-through* and the *pipelined*. The flow-through synchronous SRAM does not have a Data output register, so the output data flow asynchronously to the data I/O lines through the output buffers. The **pipelined** synchronous SRAM has a Data output register, as shown in Figure 10–12, so the output data are synchronously placed on the data I/O lines.


Burst control
Burst logic
A'_0
A'_1
A_0
A_1
CLK
A_0–A_{14} (external address)
15
Address register
15
13
Address decoder
Memory array 32k×8
8
8
WE
Write register
Data I/O control
Data input register
Data output register
CS
Enable register
Output buffers
OE
8
I/O_0–I/O_7 (Data I/O)
8
8

Data output register is in the pipelined synchronous SRAM. There is no Data output register in the flow-through synchronous SRAM.

▲ **FIGURE 10–12**

A basic block diagram of a synchronous burst SRAM

The Burst Feature

As shown in Figure 10–12, synchronous SRAMs normally have an address burst feature, which allows the memory to read or write at up to four locations using a single address. When an external address is latched in the address register, the two lowest-order address bits, A_0 and A_1, are applied to the burst logic. This produces a sequence of four internal addresses by adding 00, 01, 10, and 11 to the two lowest-order address bits on successive clock pulses. The sequence always begins with the base address, which is the external address held in the address register.

Burst Logic The address burst logic in a typical synchronous SRAM consists of a binary counter and exclusive-OR gates, as shown in Figure 10–13. For 2-bit burst logic, the internal burst address sequence is formed by the base address bits A_2–A_{14} plus the two burst address bits A'_1 and A'_0.

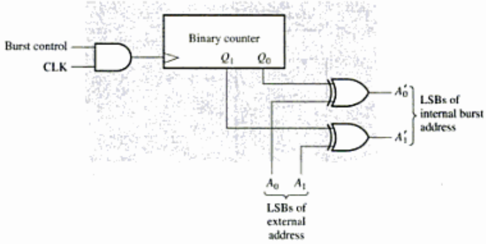
To begin the burst sequence, the counter is in its 00 state and the two lowest-order address bits are applied to the inputs of the XOR gates. Assuming that A_0 and A_1 are both 0, the internal address sequence in terms of its two lowest-order bits is 00, 01, 10, and 11.

Cache Memory

One of the major applications of SRAMs is in cache memories in computers. **Cache memory** is a relatively small, high-speed memory that stores the most recently used instructions or data from the larger but slower main memory. Cache memory uses dynamic RAM (DRAM), which

FIGURE 10–13

Address burst logic



is covered next. Typically, SRAM is several times faster than DRAM. Overall, a cache memory gets stored information to the microprocessor much faster than if only high-capacity DRAM is used. Cache memory is basically a cost-effective method of improving system performance without having to resort to the expense of making all of the memory faster.

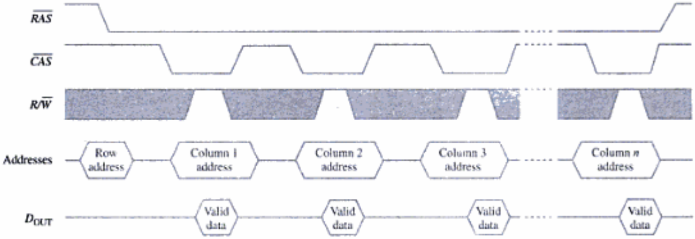
The concept of cache memory is based on the idea that computer programs tend to get instructions or data from one area of main memory before moving to another area. Basically, the cache controller "guesses" which area of the slow dynamic memory the CPU (central-processing unit) will need next and moves it to the cache memory so that it is ready when needed. If the cache controller guesses right, the data are immediately available to the microprocessor. If the cache controller guesses wrong, the CPU must go to the main memory and wait much longer for the correct instructions or data. Fortunately, the cache controller is right most of the time.

Cache Analogy There are many analogies that can be used in describing a cache memory, but comparing it to a home refrigerator is perhaps the most effective. A home refrigerator can be thought of as a "cache" for certain food items while the supermarket is the main memory where all foods are kept. Each time you want something to eat or drink, you can go to the refrigerator (cache) first to see if the item you want is there. If it is there, you save a lot of time. If it is not there, then you have to spend extra time to get it from the supermarket (main memory).

L1 and L2 Caches A first-level cache (L1 cache) is usually integrated into the processor chip and has a very limited storage capacity. L1 cache is also known as *primary* cache. A second-level cache (L2 cache) is a separate memory chip or set of chips external to the processor and usually has a larger storage capacity than an L1 cache. L2 cache is also known as *secondary* cache. Some systems may have higher-level caches (L3, L4, etc.), but L1 and L2 are the most common. Also, some systems use a disk cache to enhance the performance of the hard disk because DRAM, although much slower than SRAM, is much faster than the hard disk drive. Figure 10–14 illustrates L1 and L2 cache memories in a computer system.

Dynamic RAMs (DRAMs) Storage Cells

Dynamic memory cells store a data bit in a small capacitor rather than in a latch. The advantage of this type of cell is that it is very simple, thus allowing very large memory arrays to be constructed on a chip at a lower cost per bit. The disadvantage is that the storage capacitor cannot hold its charge over an extended period of time and will lose the stored data bit unless its charge is refreshed periodically. To refresh requires additional memory circuitry and complicates the operation of the **DRAM**. Figure 10–15 shows a typical DRAM cell consisting of a single MOS transistor (MOSFET) and a capacitor.



▲ **FIGURE 10–20**

Fast page mode timing for a read operation

In distributed refresh, each row is refreshed at intervals interspersed between normal read or write cycles. For example, the memory in Figure 10–17 has 1024 rows. As an example, for an 8 ms refresh period, each row must be refreshed every 8 ms/1024 = 7.8 μs when distributed refresh is used.

The two types of refresh operations are *$\overline{RAS}$-only refresh* and *$\overline{CAS}$ before $\overline{RAS}$ refresh*. $\overline{RAS}$ only refresh consists of a $\overline{RAS}$ transition to the LOW (active) state, which latches the address of the row to be refreshed while $\overline{CAS}$ remains HIGH (inactive) throughout the cycle. An external counter is used to provide the row addresses for this type of operation.

The $\overline{CAS}$ before $\overline{RAS}$ refresh is initiated by $\overline{CAS}$ going LOW before $\overline{RAS}$ goes LOW. This sequence activates an internal refresh counter that generates the row address to be refreshed. This address is switched by the data selector into the row decoder.

Types of DRAMs

Now that you have learned the basic concept of a DRAM, let us briefly look at the major types. These are the *Fast Page Mode (FPM) DRAM*, the *Extended Data Output (EDO) DRAM*, the *Burst Extended Data Output (BEDO) DRAM*, and the *Synchronous (S) DRAM*.

FPM DRAM Fast page mode operation was described earlier. This type of DRAM traditionally has been the most common and has been the type used in computers until the development of the EDO DRAM. Recall that a page in memory is all of the column addresses contained within one row address.

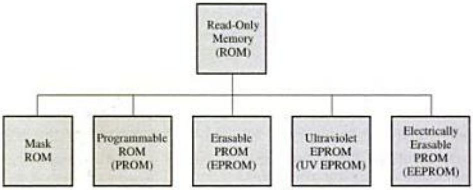
As you have seen, the basic idea of the **FPM DRAM** is based on the probability that the next several memory addresses to be accessed are in the same row (on the same page). Fortunately, this happens a large percentage of the time. FPM saves time over pure random accessing because in FPM the row address is specified only once for access to several successive column addresses whereas for pure random accessing, a row address is specified for each column address.

Recall that in a fast page mode read operation, the $\overline{CAS}$ signal has to wait until the valid data from a given address are accepted (latched) by the external system (CPU) before it can go to its nonasserted state. When $\overline{CAS}$ goes to its nonasserted state, the data outputs are disabled. This means that the next column address cannot occur until after the data from the current column address are transferred to the CPU. This limits the rate at which the columns within a page can be addressed.

EDO DRAM The Extended Data Output DRAM, sometimes called *hyper page mode DRAM*, is very similar to the FPM DRAM. The key difference is that the $\overline{CAS}$ signal in the **EDO DRAM** does not disable the output data when it goes to its nonasserted state because

FIGURE 10–21

The ROM family



by exposure to ultraviolet light over a period of several minutes. The electrically erasable PROM (**EEPROM** or E^{2}PROM) can be erased in a few milliseconds.

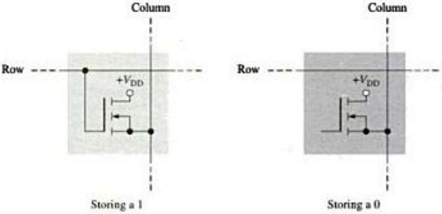
The Mask ROM

The mask ROM is usually referred to simply as a ROM. It is permanently programmed during the manufacturing process to provide widely used standard functions, such as popular conversions, or to provide user-specified functions. Once the memory is programmed, it cannot be changed. Most IC ROMs utilize the presence or absence of a transistor connection at a row/column junction to represent a 1 or a 0.

Figure 10–22 shows MOS ROM cells. The presence of a connection from a row line to the gate of a transistor represents a 1 at that location because when the row line is taken HIGH, all transistors with a gate connection to that row line turn on and connect the HIGH (1) to the associated column lines. At row/column junctions where there are no gate connections, the column lines remain LOW (0) when the row is addressed.

FIGURE 10–22

ROM cells



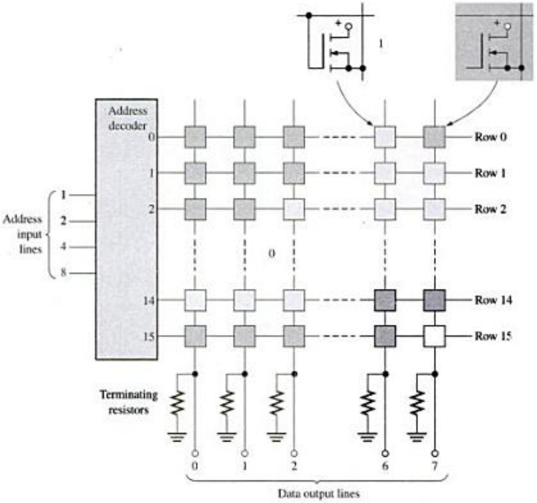
A Simple ROM

To illustrate the ROM concept, Figure 10–23 shows a small, simplified ROM array. The unshaded squares represent stored 1s, and the shaded squares represent stored 0s. The basic read operation is as follows: When a binary address code is applied to the address input lines, the corresponding row line goes HIGH. This HIGH is connected to the column lines through the transistors at each junction (cell) where a 1 is stored. At each cell where a 0 is stored, the column line stays LOW because of the terminating resistor. The column lines form the data output. The eight data bits stored in the selected row appear on the output lines.

As you can see, the example ROM in Figure 10–23 is organized into 16 addresses, each of which stores 8 data bits. Thus, it is a 16 × 8 (16-by-8) ROM, and its total capacity is 128 bits or 16 bytes. ROMs can be used as look-up tables (LUTs) for code conversions and logic function generation. LUTs are discussed later in the chapter.

FIGURE 10–23

A 16 × 8-bit ROM array


1
Address decoder
0
1
2
14
15
Address input lines
1
2
4
8
Row 0
Row 1
Row 2
0
Row 14
Row 15
Terminating resistors
0
1
2
6
7
Data output lines

EXAMPLE 10–1

Show a basic ROM, similar to the one in Figure 10–23, programmed for a 4-bit binary-to-Gray conversion.

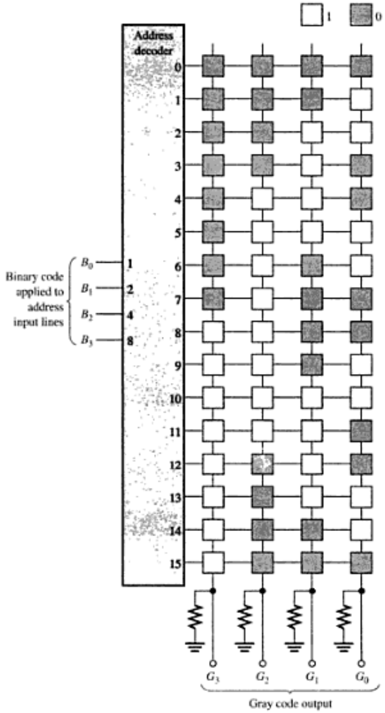
Solution Review Chapter 2 for the Gray code. Table 10–1 is developed for use in programming the ROM.

TABLE 10–1

BINARY				GRAY			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

The resulting 16 × 4 ROM array is shown in Figure 10–24. You can see that a binary code on the address input lines produces the corresponding Gray code on the output lines (columns). For example, when the binary number 0110 is applied to the address input lines, address 6, which stores the Gray code 0101, is selected.

▶ **FIGURE 10–24**


1
0
Address decoder
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Binary code applied to address input lines
B_0 1
B_1 2
B_2 4
B_3 8
G_3
G_2
G_1
G_0
Gray code output

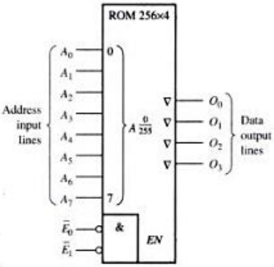
Supplementary Problem Using Figure 10–24, determine the Gray code output when a binary code of 1011 is applied to the address input lines.

Internal ROM Organization

Most IC ROMs have a more complex internal organization than that in the basic simplified example just presented. To illustrate how an IC ROM is structured, let us use a 1024-bit device with a 256 × 4 organization. The logic symbol is shown in Figure 10–25. When any one of 256 binary codes (eight bits) is applied to the address lines, four data bits appear on the outputs if the chip enable inputs are LOW. (There are eight address lines because $2^8 = 256$.) The A_{255}^{0} designator means that the 8-bit address code selects address 0 through 255.

▶ FIGURE 10–25

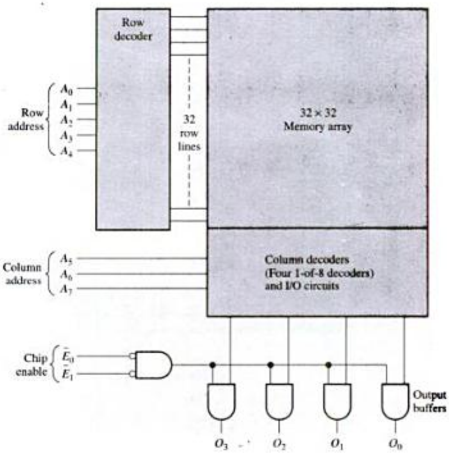
A 256 × 4 ROM logic symbol


ROM 256×4
Address input lines
A_0
A_1
A_2
A_3
A_4
A_5
A_6
A_7
0
7
$A\frac{0}{255}$
O_0
O_1
O_2
O_3
Data output lines
$\bar{E}_0$
$\bar{E}_1$
&
EN

Although the 256 × 4 organization of this device implies that there are 256 rows and 4 columns in the memory array, this is not actually the case. The memory cell array is actually a 32 × 32 matrix (32 rows and 32 columns), as shown in the block diagram in Figure 10–26.

▶ FIGURE 10–26

A typical 1024-bit ROM


Row decoder
Row address
A_0
A_1
A_2
A_3
A_4
32 row lines
32 × 32 Memory array
Column address
A_5
A_6
A_7
Column decoders (Four 1-of-8 decoders) and I/O circuits
Chip enable
$\bar{E}_0$
$\bar{E}_1$
Output buffers
O_3
O_2
O_1
O_0

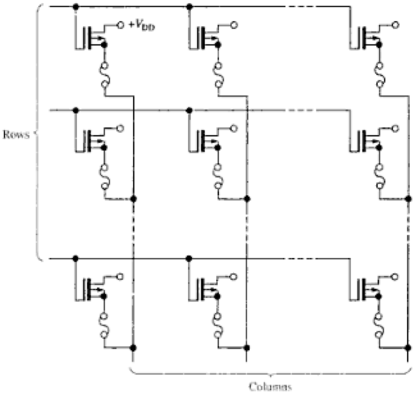
PROMs

A **PROM** uses some type of fusing process to store bits, in which a memory *link* is burned open or left intact to represent a 0 or a 1. The fusing process is irreversible; once a PROM is programmed, it cannot be changed.

Figure 10–28 illustrates a MOS PROM array with fusible links. The fusible links are manufactured into the PROM between the source of each cell's transistor and its column line. In the programming process, a sufficient current is injected through the fusible link to burn it open to create a stored 0. The link is left intact for a stored 1. In Figure 10–28, all drains are commonly connected to V_{DD}.

FIGURE 10–28

MOS PROM array with fusible links



Three basic fuse technologies used in PROMs are metal links, silicon links, and *pn* junctions. A brief description of each of these are as follows:

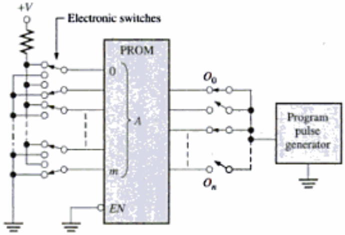
1. Metal links are made of a material such as nichrome. Each bit in the memory array is represented by a separate link. During programming, the link is either "blown" open or left intact. This is done basically by first addressing a given cell and then forcing a sufficient amount of current through the link to cause it to open.
2. Silicon links are formed by narrow, notched strips of polycrystalline silicon. Programming of these fuses requires melting of the links by passing a sufficient amount of current through them. This amount of current causes a high temperature at the fuse location that oxidizes the silicon and forms an insulation around the now-open link.
3. Shorted junction, or avalanche-induced migration, technology consists basically of two *pn* junctions arranged back-to-back. During programming, one of the diode junctions is avalanched, and the resulting voltage and heat cause aluminum ions to migrate and short the junction. The remaining junction is then used as a forward-biased diode to represent a data bit.

PROM Programming

A PROM is normally programmed by inserting it into a special instrument called a *PROM programmer*. Basically, the programming is accomplished as shown by the simplified setup in Figure 10–29. An address is selected by the electronic switches on the address lines, and then a pulse is applied to those output lines corresponding to bit locations where 0s are to be stored (the PROM starts out with all 1s). These pulses blow the fusible links, thus creating the desired bit pattern. The next address is then selected and the process is repeated. This sequence is actually done automatically by a software-driven PROM programmer.

FIGURE 10–29

Simplified concept of PROM programming setup



EPROMs

An **EPROM** is an erasable PROM. Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased first.

An EPROM uses an NMOSFET array with an isolated-gate structure. The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time. The data bits in this type of array are represented by the presence or absence of a stored gate charge. Erasure of a data bit is a process that removes the gate charge.

Two basic types of erasable PROMs are the ultraviolet erasable PROM (UV EPROM) and the electrically erasable PROM (EEPROM).

UV EPROMs You can recognize the UV EPROM device by the transparent quartz lid on the package, as shown in Figure 10–30. The isolated gate in the **FET** of an ultraviolet EPROM is "floating" within an oxide insulating material. The programming process causes electrons to be removed from the floating gate. Erasure is done by exposure of the memory array chip to high-intensity ultraviolet radiation through the quartz window on top of the package. The positive charge stored on the gate is neutralized after several minutes to an hour of exposure time.

FIGURE 10–30

Ultraviolet erasable PROM package

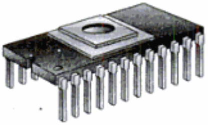
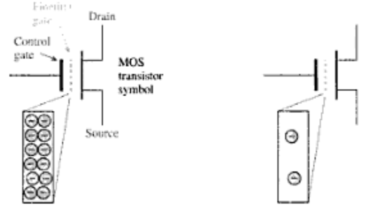


FIGURE 10–33

The storage cell in a flash memory



Many electrons = more charge = stored 0.

Few electrons = less charge = stored 1.

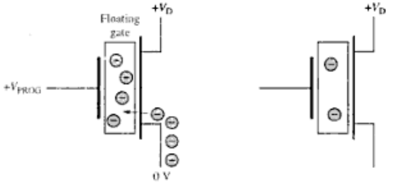
Basic Flash Memory Operation

There are three major operations in a flash memory: the *programming* operation, the *read* operation, and the *erase* operation.

Programming Initially, all cells are at the 1 state because charge was removed from each cell in a previous erase operation. The programming operation adds electrons (charge) to the floating gate of those cells that are to store a 0. No charge is added to those cells that are to store a 1. Application of a sufficient positive voltage to the control gate with respect to the source during programming attracts electrons to the floating gate, as indicated in Figure 10–34. Once programmed, a cell can retain the charge for up to 100 years without any external power.

FIGURE 10–34

Simplified illustration of storing a 0 or a 1 during the programming operation



To store a 0, a sufficient positive voltage is applied to the control gate with respect to the source to add charge to the floating gate during programming.

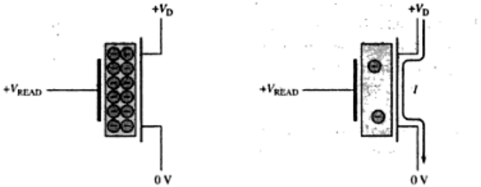
To store a 1, no charge is added and the cell is left in the erased condition.

Read During a read operation, a positive voltage is applied to the control gate. The amount of charge present on the floating gate of a cell determines whether or not the voltage applied to the control gate will turn on the transistor. If a 1 is stored, the control gate voltage is sufficient to turn the transistor on. If a 0 is stored, the transistor will not turn on because the control gate voltage is not sufficient to overcome the negative charge stored in the floating gate. Think of the charge on the floating gate as a voltage source that opposes the voltage applied to the control gate during a read operation. So, the floating gate charge associated with a stored 0 prevents the control gate voltage from reaching the turn-on threshold, whereas the small or zero charge associated with a stored 1 allows the control gate voltage to exceed the turn-on threshold.

When the transistor turns on, there is current from the drain to the source of the cell transistor. The presence of this current is sensed to indicate a 1, and the absence of this current is sensed to indicate a 0. This basic idea is illustrated in Figure 10–35.

▶ FIGURE 10–35

The read operation of a flash cell in an array



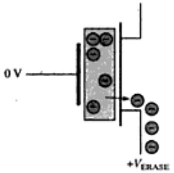
When a 0 is read, the transistor remains off because the charge on the floating gate prevents the read voltage from exceeding the turn-on threshold.

When a 1 is read, the transistor turns on because the absence of charge on the floating gate allows the read voltage to exceed the turn-on threshold.

Erase During an erase operation, charge is removed from all the memory cells. A sufficient positive voltage is applied to the transistor source with respect to the control gate. This is opposite in polarity to that used in programming. This voltage attracts electrons from the floating gate and depletes it of charge, as illustrated in Figure 10–36. A flash memory is always erased prior to being reprogrammed.

▶ FIGURE 10–36

Simplified illustration of removing charge from a cell during erase



To erase a cell, a sufficient positive voltage is applied to the source with respect to the control gate to remove charge from the floating gate during the erase operation.

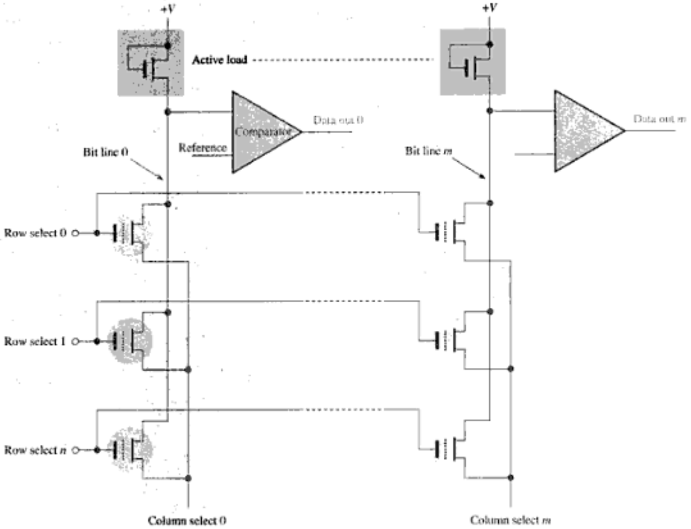
Basic Flash Memory Array

A simplified array of flash memory cells is shown in Figure 10–37. Only one row line is accessed at a time. When a cell in a given bit line turns on (stored 1) during a read operation, there is current through the bit line, which produces a voltage drop across the active load. This voltage drop is compared to a reference voltage with a comparator circuit and an output level indicating a 1 is produced. If a 0 is stored, then there is no current or a little current in the bit line and an opposite level is produced on the comparator output.

Comparison of Flash Memories with Other Memories

Let us compare flash memories with other types of memories with which you are already familiar.

Flash vs. ROM, EPROM, and EEPROM Read-only memories are high-density, nonvolatile devices. However, once programmed the contents of a ROM can never be altered. Also, the initial programming is a time-consuming and costly process.



▲ **FIGURE 10–37**

Basic flash memory array

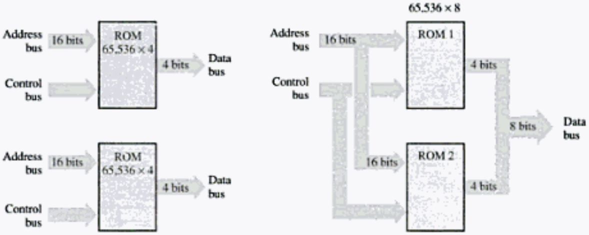
Although the EPROM is a high-density, nonvolatile memory, it can be erased only by removing it from the system and using ultraviolet light. It can be reprogrammed only with specialized equipment.

The EEPROM has a more complex cell structure than either the ROM or EPROM and so the density is not as high, although it can be reprogrammed without being removed from the system. Because of its lower density, the cost/bit is higher than ROMs or EPROMs.

A flash memory can be reprogrammed easily in the system because it is essentially a READ/WRITE device. The density of a flash memory compares with the ROM and EPROM because both have single transistor cells. A flash memory (like a ROM, EPROM, or EEPROM) is nonvolatile, which allows data to be stored indefinitely with power off.

Flash vs. SRAM As you have learned, static random-access memories are volatile READ/WRITE devices. A SRAM requires constant power to retain the stored data. In many applications, a battery backup is used to prevent data loss if the main power source is turned off. However, since battery failure is always a possibility, indefinite retention of the stored data in a SRAM cannot be guaranteed. Because the memory cell in a SRAM is basically a flip-flop consisting of several transistors, the density is relatively low.

A flash memory is also a READ/WRITE memory, but unlike the SRAM it is nonvolatile. Also, a flash memory has a much higher density than a SRAM.


Address bus
16 bits
ROM 65,536 × 4
4 bits
Data bus
Control bus
Address bus
16 bits
ROM 65,536 × 4
4 bits
Data bus
Control bus
65,536 × 8
Address bus
16 bits
ROM 1
4 bits
Control bus
8 bits
Data bus
16 bits
ROM 2
4 bits

(a) Two separate 65,536 × 4 ROMs

(b) One 65,536 × 8 ROM from two 65,536 × 4 ROMs

▲ **FIGURE 10–38**

Illustration of word-length expansion

address bus is commonly connected to both memories so that the combination memory still has the same number of addresses ($2^{16} = 65{,}536$) as each individual memory. The 4-bit data buses from the two memories are combined to form an 8-bit data bus. Now, when an address is selected, eight bits are produced on the data bus—four from each memory.

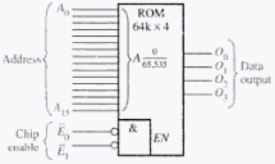
The following example shows the details of 65,536 × 4 to 65,536 × 8 expansion.

EXAMPLE 10–2

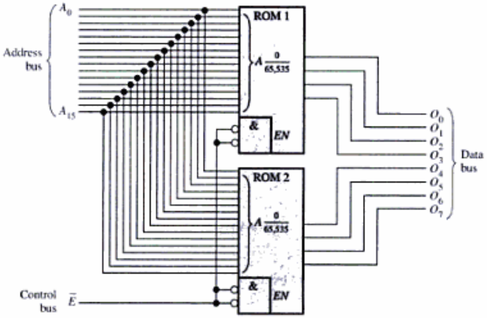
Expand the 65,536 × 4 ROM (64k × 4) in Figure 10–39 to form a 64k × 8 ROM. Note that "64k" is the accepted shorthand for 65,536. Why not "65k"? Maybe, it is because 64 is also a power-of-two.

▶ **FIGURE 10–39**

A 64k × 4 ROM


ROM 64k × 4
A_0
Address
A_{15}
$A \frac{0}{65,535}$
O_0
O_1
O_2
O_3
Data output
Chip enable
$\bar{E}_0$
$\bar{E}_1$
&
EN

Solution Two 64k × 4 ROMs are connected as shown in Figure 10–40. Notice that a specific address is accessed in ROM 1 and ROM 2 at the same time. The four bits from a selected address in ROM 1 and the four bits from the corresponding address in ROM 2 go out in parallel to form an 8-bit word on the data bus. Also, notice that a LOW on the chip enable line, $\bar{E}$, which forms a simple control bus, enables *both* memories.



A_0
Address bus
A_{15}
ROM 1
$A\frac{0}{65,535}$
&
EN
ROM 2
$A\frac{0}{65,535}$
&
EN
O_0
O_1
O_2
O_3
O_4
O_5
O_6
O_7
Data bus
Control bus
$\overline{E}$

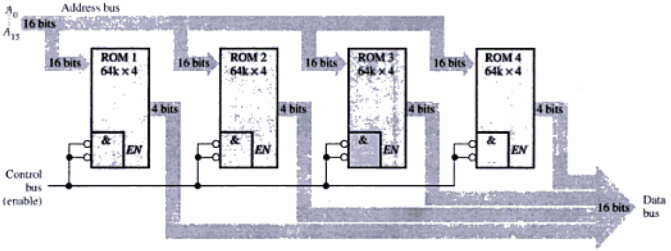
▶ **FIGURE 10–40**

Supplementary Problem Describe how you would expand a 64k × 1 ROM to a 64k × 8 ROM.

EXAMPLE 10–3

Use the memories in Example 10–2 to form a 64k × 16 ROM.

Solution In this case you need a memory that stores 65,536 16-bit words. Four 64k × 4 ROMs are required to do the job, as shown in Figure 10–41.



Address bus
A_0
A_{15}
16 bits
16 bits
ROM 1 64k × 4
16 bits
ROM 2 64k × 4
16 bits
ROM 3 64k × 4
16 bits
ROM 4 64k × 4
4 bits
4 bits
4 bits
4 bits
&
EN
Control bus (enable)
16 bits
Data bus

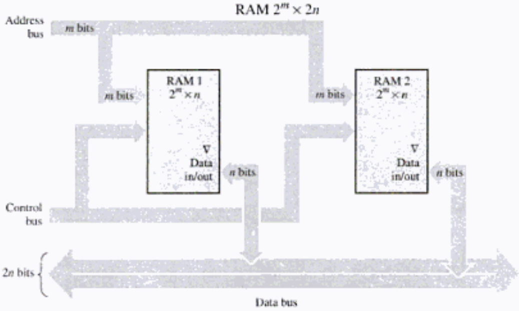
▲ **FIGURE 10–41**

Supplementary Problem How many 64k × 1 ROMs would be required to implement the memory shown in Figure 10–41?

A ROM has only data outputs, but a RAM has both data inputs and data outputs. For word-length expansion in a RAM (SRAM or DRAM), the data inputs *and* data outputs form the data bus. Because the same lines are used for data input and data output, tristate buffers are required. Most RAMs provide internal tristate circuitry. Figure 10–42 illustrates RAM expansion to increase the data word length.

▶ FIGURE 10–42

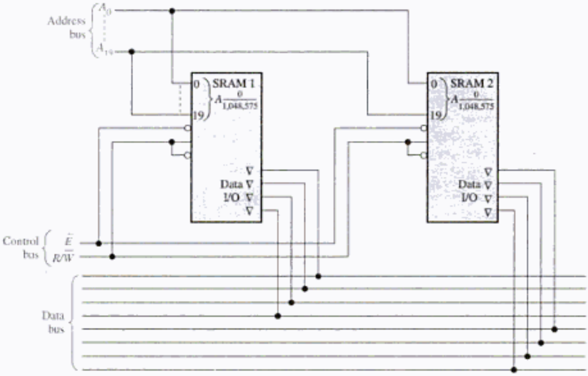
Illustration of word-length expansion


RAM $2^m \times 2n$
Address bus
m bits
m bits
m bits
RAM 1
$2^m \times n$
RAM 2
$2^m \times n$
Data in/out
Data in/out
n bits
n bits
Control bus
$2n$ bits
Data bus

EXAMPLE 10–4

Use 1M × 4 SRAMs to create a 1M × 8 SRAM.

Solution Two 1M × 4 SRAMs are connected as shown in the simplified block diagram of Figure 10–43.


Address bus
A_0
A_{19}
0
19
SRAM 1
$A \frac{0}{1{,}048{,}575}$
Data I/O
0
19
SRAM 2
$A \frac{0}{1{,}048{,}575}$
Data I/O
Control bus
$\bar{E}$
$R/\bar{W}$
Data bus

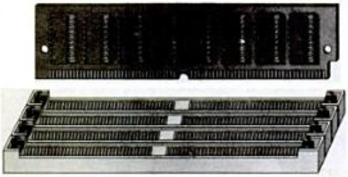
▲ FIGURE 10–43

Supplementary Problem Use 1M × 8 SRAMs to create a 1M × 16 SRAM.

SIMMs and DIMMs plug into sockets on a system board such as those illustrated in Figure 10–47 where several sockets are generally available for memory expansion. The sockets for SIMMs and DIMMs, of course, are different and not interchangeable.

▶ FIGURE 10–47

A SIMM/DIMM inserted into a socket on a system board



SECTION 10–6 REVIEW

1. How many 16k × 1 RAMs are required to achieve a memory with a word capacity of 16k and a word length of eight bits?
2. To expand the 16k × 8 memory in question 1 to a 32k × 8 organization, how many more 16k × 1 RAMs are required?
3. What does SIMM stand for?
4. What does DIMM stand for?

10–7 SPECIAL TYPES OF MEMORIES

In this section, the first in–first out (FIFO) memory, the last in–first out (LIFO) memory, the memory stack, and the charge-coupled device memory are covered.

After completing this section, you should be able to

- Describe a FIFO memory
- Describe a LIFO memory
- Discuss memory stacks
- Explain how to use a portion of RAM as a memory stack
- Describe a basic CCD memory

First In–First Out (FIFO) Memories

This type of memory is formed by an arrangement of shift registers. The term **FIFO** refers to the basic operation of this type of memory, in which the first data bit written into the memory is the first to be read out.

One important difference between a conventional shift register and a FIFO register is illustrated in Figure 10–48. In a conventional register, a data bit moves through the register only as new data bits are entered; in a FIFO register, a data bit immediately goes through the register to the right-most bit location that is empty.

▶ FIGURE 10–48

Conventional shift register

Input	*X*	*X*	*X*	*X*	Output
0	0	*X*	*X*	*X*	→
1	1	0	*X*	*X*	→
1	1	1	0	*X*	→
0	0	1	1	0	→

X = unknown data bits.

In a conventional shift register, data stay to the left until "forced" through by additional data.

FIFO shift register

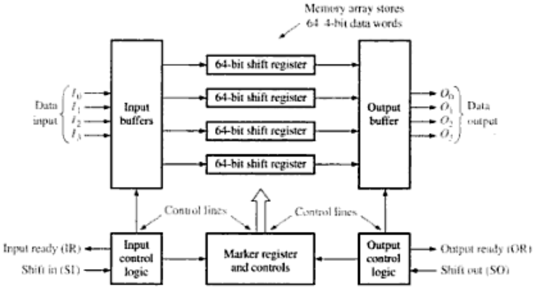
Input	—	—	—	—	Output
0	—	—	—	0	→
1	—	—	1	0	→
1	—	1	1	0	→
0	0	1	1	0	→

— = empty positions.

In a FIFO shift register, data "fall" through (go right).

Figure 10–49 is a block diagram of a FIFO serial memory. This particular memory has four serial 64-bit data registers and a 64-bit control register (marker register). When data are entered by a shift-in pulse, they move automatically under control of the marker register to the empty location closest to the output. Data cannot advance into occupied positions. However, when a data bit is shifted out by a shift-out pulse, the data bits remaining in the registers automatically move to the next position toward the output. In an asynchronous FIFO, data are shifted out independent of data entry, with the use of two separate clocks.

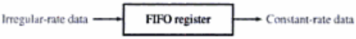
▶ FIGURE 10–49


Memory array stores 64 4-bit data words
64-bit shift register
64-bit shift register
64-bit shift register
64-bit shift register
Data input
I_0
I_1
I_2
I_3
Input buffers
Output buffer
O_0
O_1
O_2
O_3
Data output
Control lines
Control lines
Input ready (IR)
Shift in (SI)
Input control logic
Marker register and controls
Output control logic
Output ready (OR)
Shift out (SO)

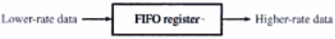
FIFO Applications

One important application area for the FIFO register is the case in which two systems of differing data rates must communicate. Data can be entered into a FIFO register at one rate and taken out at another rate. Figure 10–50 illustrates how a FIFO register might be used in these situations.

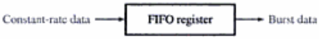
FIGURE 10–50


Irregular-rate data
FIFO register
Constant-rate data

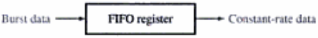
(a) Irregular telemetry data can be stored and retransmitted at a constant rate.


Lower-rate data
FIFO register
Higher-rate data

(b) Data input at a slow keyboard rate can be stored and then transferred at a higher rate for processing.


Constant-rate data
FIFO register
Burst data

(c) Data input at a constant rate can be stored and then output in bursts.


Burst data
FIFO register
Constant-rate data

(d) Data in bursts can be stored and reformatted into a constant-rate output.

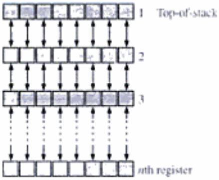
Last In–First Out (LIFO) Memories

The **LIFO** (last in–first out) memory is found in applications involving microprocessors and other computing systems. It allows data to be stored and then recalled in reverse order; that is, the last data byte to be stored is the first data byte to be retrieved.

Register Stacks A LIFO memory is commonly referred to as a push-down stack. In some systems, it is implemented with a group of registers as shown in Figure 10–51. A stack can consist of any number of registers, but the register at the top is called the *top-of-stack.*

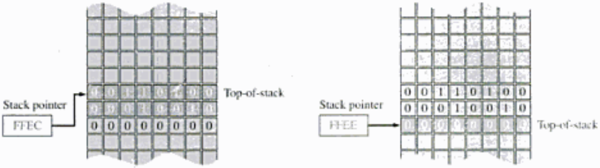
FIGURE 10–51

Register stack


1 Top-of-stack
2
3
nth register

To illustrate the principle, a byte of data is loaded in parallel onto the top of the stack. Each successive byte pushes the previous one down into the next register. This process is illustrated in Figure 10–52. Notice that the new data byte is always loaded into the top register and the previously stored bytes are pushed deeper into the stack. The name *push-down stack* comes from this characteristic.

Data bytes are retrieved in the reverse order. The last byte entered is always at the top of the stack, so when it is pulled from the stack, the other bytes pop up into the next higher locations. This process is illustrated in Figure 10–53.



(a) The stack pointer is at FFEC before the data word is copied (popped) from the stack.

(b) The stack pointer is incremented by two and the last data word stored is copied (popped) from the stack.

▲ **FIGURE 10–56**

CCD Memories

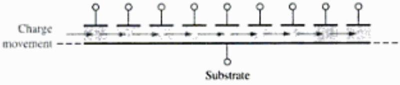
The **CCD** (charge-coupled device) memory stores data as charges on capacitors. Unlike the dynamic RAM, however, the storage cell does not include a transistor. High density is the main advantage of CCDs.

The CCD memory consists of long rows of semiconductor capacitors, called *channels*. Data are entered into a channel serially by depositing a small charge for a 0 and a large charge for a 1 on the capacitors. These charge packets are then shifted along the channel by clock signals as more data are entered.

As with the DRAM, the charges must be refreshed periodically. This process is done by shifting the charge packets serially through a refresh circuit. Figure 10–57 shows the basic concept of a CCD channel. Because data are shifted serially through the channels, the CCD memory has a relatively long access time. CCD arrays are used in some modern cameras to capture video images in the form of light-induced charge.

▶ **FIGURE 10–57**

A CCD (charge-coupled device) channel



SECTION 10–7 REVIEW

1. What is a FIFO memory?
2. What is a LIFO memory?
3. Explain the PUSH operation in a memory stack.
4. Explain the POP operation in a memory stack.
5. What does the term CCD stand for?

10-8 MAGNETIC AND OPTICAL STORAGE

In this section, you will be introduced to the basics of magnetic disks, magnetic tape, magneto-optical disks, and optical disks. These storage media are very important, particularly in computer applications, where they are used for mass nonvolatile storage of data and programs.

After completing this section, you should be able to

■ Describe a magnetic hard disk ■ Describe a floppy disk ■ Discuss removable hard disks ■ Explain the principle of magneto-optical disks ■ Discuss the CD-ROM, CD-R, and CD-RW disks ■ Describe the WORM ■ Discuss the DVD-ROM

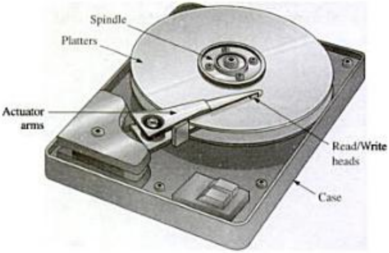
Magnetic Storage

Magnetic Hard Disks Computers use hard disks as the internal mass storage media. **Hard disks** are rigid "platters" made of aluminum alloy or a mixture of glass and ceramic covered with a magnetic coating. Hard disk drives mainly come in two diameter sizes, 5.25 in. and 3.5 in. although 2.5 in. and 1.75 in. are also available. A hard disk drive is hermetically sealed to keep the disks dust-free.

Typically, two or more platters are stacked on top of each other on a common shaft or spindle that turns the assembly at several thousand rpm. There is a separation between each disk to allow for a magnetic read/write head that is mounted on the end of an actuator arm, as shown in Figure 10–58. There is a read/write head for both sides of each disk since data are recorded on both sides of the disk surface. The drive actuator arm synchronizes all the read/write heads to keep them in perfect alignment as they "fly" across the disk surface with a separation of only a fraction of a millimetre from the disk. A small dust particle could cause a head to "crash," causing damage to the disk surface.

► **FIGURE 10–58**

A hard disk drive



Basic Read/Write Head Principles The hard drive is a random-access device because it can retrieve stored data anywhere on the disk in any order. A simplified diagram of the magnetic surface read/write operation is shown in Figure 10–59. The direction or polarization of the magnetic domains on the disk surface is controlled by the direction of the magnetic flux lines (magnetic field) produced by the write head according to the direction of a current pulse in the winding. This magnetic flux magnetizes a small spot on the disk surface in the direction of the magnetic field. A magnetized spot of one polarity represents a binary 1, and one of the opposite polarity represents a binary 0. Once a spot on the disk surface is magnetized, it remains until written over with an opposite magnetic field.

Hard Disk Performance Several basic parameters determine the performance of a givei hard disk drive. A *seek* operation is the movement of the read/write head to the desired track. The **seek time** is the average time for this operation to be performed. Typically, hard disk drives have an average seek time of several milliseconds, depending on the particular drive.

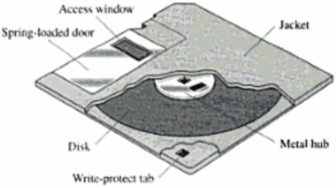
The **latency period** is the time it takes for the desired sector to spin under the head once the head is positioned over the desired track. A worst case is when the desired sector is just past the head position and spinning away from it. The sector must rotate almost a full revolution back to the head position. *Average latency period* assumes that the disk must make half of a revolution. Obviously, the latency period depends on the constant rotational speed of the disk. Disk rotation speeds are different for different disk drives but typically are 3600 rpm, 4500 rpm, 5400 rpm, and 7200 rpm. Some disk drives rotate at 10,033 rpm and have an average latency period of less than 3 ms.

The sum of the average seek time and the average latency period is the *access time* for the disk drive.

Floppy Disks The floppy disk is an older technology and derives its name because it is made of a flexible polyester material with a magnetic coating on both sides. The early floppy disks were 5.25 inches in diameter and were packaged in a semiflexible jacket. Current **floppy disks** or diskettes are 3.5 inches in diameter and are encased in a rigid plastic jacket, as shown in Figure 10–61. A spring-loaded shutter covers the access window and remains closed until the disk is inserted into a disk drive. A metal hub has one hole to centre the disk and another for spinning it within the protective jacket. Obviously, floppy disks are removable disks, whereas hard disks are not. Floppy disks are formatted into tracks and sectors similar to hard disks except for the number of tracks and sectors. The high-density 1.44 Mbyte floppies have 80 tracks per side with 18 sectors. With the advent of other types of removable disks such as Zip, the floppy has serious competition, but its low cost may continue to make the floppy disk competitive for smaller storage applications.

▶ FIGURE 10–61

The 3.5 inch floppy disk (diskette)

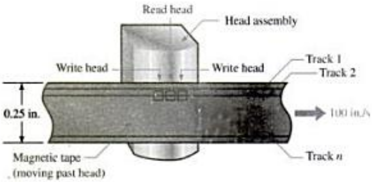


Zip™ The Zip drive is one type of removable magnetic storage device that has replaced the limited-capacity floppy. Like the floppy disk, the **Zip disk** cartridge is a flexible disk housed in a rigid case about the same size as that of the floppy disk but thicker. The typical Zip drive is much faster than the floppy drive because it has a 3000 rpm spin rate compared to the floppy's 300 rpm. The Zip drive has a storage capacity of 100 Mbytes, about 69 times more than the 1.44 Mbyte floppy. A typical external Zip drive and cartridge are shown in Figure 10–62.

Jaz™ Another type of removable magnetic storage device is the Jaz drive, which is similar to a hard disk drive except that two platters are housed in a removable cartridge protected by a dust-proof shutter. The **Jaz cartridges** are available with storage capacities of 1 or 2 Gbytes. A Jaz drive and cartridge are shown in Figure 10–63.

▶ **FIGURE 10–64**

QIC tape



DLT is an abbreviation for digital linear tape. DLT is a half-inch wide tape, which is 60% wider than 8 mm and, of course, twice as wide as standard QIC. Basically, DLT differs in the way the tape-drive mechanism works to minimize tape wear compared to other systems. DLT offers the highest storage capacity of all the tape formats with capacities ranging up to 35 Gbytes.

Magneto-Optical Storage

As the name implies, magneto-optical (MO) storage devices use a combination of magnetic and optical (laser) technologies. A **magneto-optical disk** is formatted into tracks and sectors similar to magnetic disks.

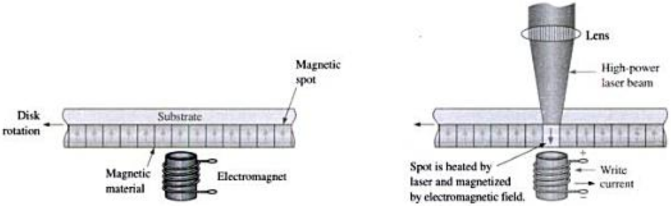
The basic difference between a purely magnetic disk and an MO disk is that the magnetic coating used on the MO disk requires heat to alter the magnetic polarization. Therefore, the MO is extremely stable at ambient temperature, making data unchangeable. To write a data bit, a high-power laser beam is focused on a tiny spot on the disk, and the temperature of that tiny spot is raised above a temperature level called the Curie point (about 200°C). Once heated, the magnetic particles at that spot can easily have their direction (polarization) changed by a magnetic field generated by the write head. Information is read from the disk with a less-powerful laser than used for writing, making use of the Kerr effect where the polarity of the reflected laser light is altered depending on the orientation of the magnetic particles. Spots of one polarity represent 0s and spots of the opposite polarity represent 1s. Basic MO operation is shown in Figure 10–65, which represents a small cross-sectional area of a disk.

Optical Storage

CD-ROM The basic Compact Disk-Read-Only Memory is a 120 mm diameter disk with a sandwich of three coatings: a polycarbonate plastic on the bottom, a thin aluminum sheet for reflectivity, and a top coating of lacquer for protection. The **CD-ROM** disk is formatted in a single spiral track with sequential 2 kbyte sectors and has a capacity of 680 Mbytes. Data is prerecorded at the factory in the form of minute indentations called *pits* and the flat area surrounding the pits called *lands*. The pits are stamped into the plastic layer and cannot be erased.

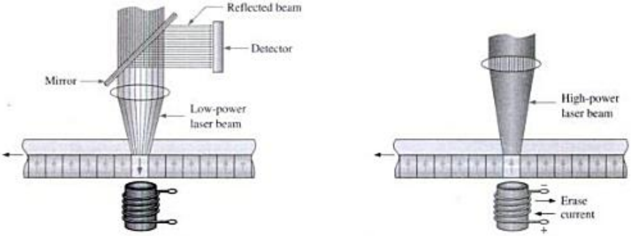
A CD player reads data from the spiral track with a low-power infrared laser, as illustrated in Figure 10–66. The data are in the form of pits and lands as shown. Laser light reflected from a pit is 180° out-of-phase with the light reflected from the lands. As the disk rotates, the narrow laser beam strikes the series of pits and lands of varying lengths, and a photodiode detects the difference in the reflected light. The result is a series of 1s and 0s corresponding to the configuration of pits and lands along the track.

WORM Write Once/Read Many (**WORM**) is a type of optical storage that can be written onto one time after which the data cannot be erased but can be read many times. To write data, a low-power laser is used to burn microscopic pits on the disk surface. 1s and 0s are represented by the burned and nonburned areas.


Magnetic spot
Disk rotation
Substrate
Magnetic material
Electromagnet
Lens
High-power laser beam
Spot is heated by laser and magnetized by electromagnetic field.
+
Write current
−

(a) Unrecorded disk

(b) Writing: A high-power laser beam heats the spot, causing the magnetic particles to align with the electromagnetic field.


Reflected beam
Detector
Mirror
Low-power laser beam
High-power laser beam
−
Erase current
+

(c) Reading: A low-power laser beam reflects off of the reversed-polarity magnetic particles and its polarization shifts. If the particles are not reversed, the polarization of the reflected beam is unchanged.

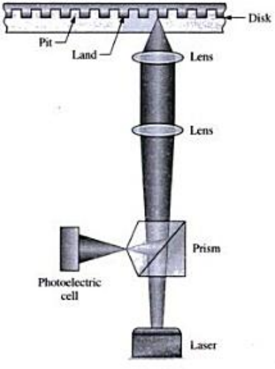
(d) Erasing: The electromagnetic field is reversed as the high-power laser beam heats the spot, causing the magnetic particles to be restored to the original polarity.

▲ **FIGURE 10–65**

Basic principle of a magneto-optical disk

► **FIGURE 10–66**

Basic operation of reading data from a CD-ROM


Disk
Pit
Land
Lens
Lens
Prism
Photoelectric cell
Laser

ROM Testing

Since ROMs contain known data, they can be checked for the correctness of the stored data by reading each data word from the memory and comparing it with a data word that is known to be correct. One way of doing this is illustrated in Figure 10–67. This process requires a reference ROM that contains the same data as the ROM to be tested. A special test instrument is programmed to read each address in both ROMs simultaneously and to compare the contents. A flowchart in Figure 10–68 illustrates the basic sequence.

FIGURE 10–67

Block diagram for a complete check of a ROM

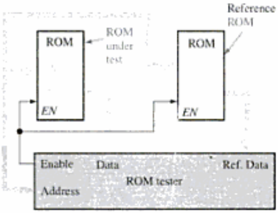
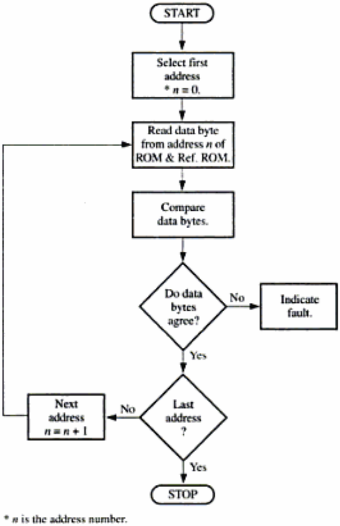

Reference ROM
ROM
ROM under test
ROM
EN
EN
Enable
Data
Ref. Data
ROM tester
Address

FIGURE 10–68

Flowchart for a complete contents check of a ROM


START
Select first address *n = 0.
Read data byte from address n of ROM & Ref. ROM.
Compare data bytes.
Do data bytes agree?
No
Indicate fault.
Yes
Last address ?
No
Next address n = n + 1
Yes
STOP
* n is the address number.

The checksum test can be implemented with a special test instrument, or it can be incorporated as a test routine in the built-in (system) software or microprocessor-based systems. In that case, the ROM test routine is automatically run on system start-up.

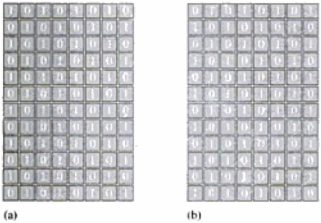
RAM Testing

To test a RAM for its ability to store both 0s and 1s in each cell, first 0s are written into all the cells in each address and then read out and checked. Next, 1s are written into all the cells in each address and then read out and checked. This basic test will detect a cell that is stuck in either a 1 state or a 0 state.

Some memory faults cannot be detected with the all-0s–all-1s test. For example, if two adjacent memory cells are shorted, they will always be in the same state, both 0s or both 1s. Also, the all-0s–all-1s test is ineffective if there are internal noise problems such that the contents of one or more addresses are altered by a change in the contents of another address.

The Checkerboard Pattern Test One way to more fully test a RAM is by using a checkerboard pattern of 1s and 0s, as illustrated in Figure 10–71. Notice that all adjacent cells have opposite bits. This pattern checks for a short between two adjacent cells; if there is a short, both cells will be in the same state.

After the RAM is checked with the pattern in Figure 10–71(a), the pattern is reversed, as shown in part (b). This reversal checks the ability of all cells to store both 1s and 0s.



▲ **FIGURE 10–71**

The RAM checkboard test pattern

A further test is to alternate the pattern one address at a time and check all the other addresses for the proper pattern. This test will catch a problem in which the contents of an address are dynamically altered when the contents of another address change.

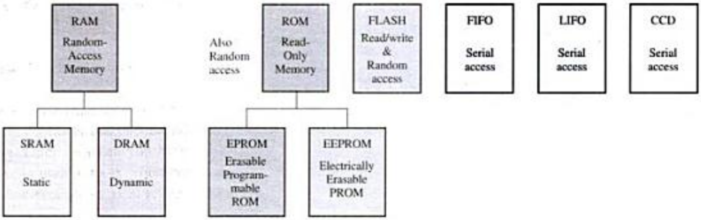
A basic procedure for the checkerboard test is illustrated by the flowchart in Figure 10–72. The procedure can be implemented with the system software in microprocessor-based systems so that either the tests are automatic when the system is powered up or they can be initiated from the keyboard.

SECTION 10–9 REVIEW

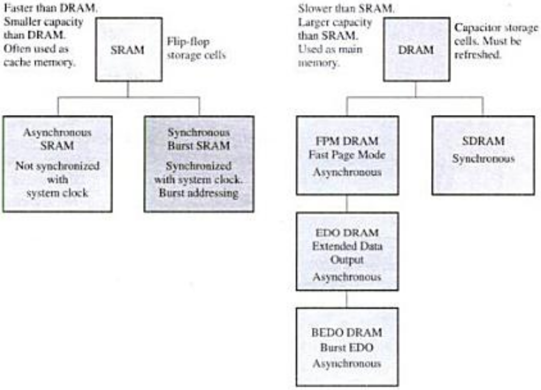
1. Describe the checksum method of ROM testing.
2. Why can the checksum method not be applied to RAM testing?
3. List the three basic faults that the checkerboard pattern test can detect in a RAM.

SUMMARY

- Types of semiconductor memories:


RAM
Random-Access Memory
SRAM
Static
DRAM
Dynamic
Also Random access
ROM
Read-Only Memory
EPROM
Erasable Programmable ROM
EEPROM
Electrically Erasable PROM
FLASH
Read/write & Random access
FIFO
Serial access
LIFO
Serial access
CCD
Serial access

- Types of SRAMs (Static RAMs) and DRAMs (Dynamic RAMs):


Faster than DRAM. Smaller capacity than DRAM. Often used as cache memory.
SRAM
Flip-flop storage cells
Asynchronous SRAM
Not synchronized with system clock
Synchronous Burst SRAM
Synchronized with system clock. Burst addressing
Slower than SRAM. Larger capacity than SRAM. Used as main memory.
DRAM
Capacitor storage cells. Must be refreshed.
FPM DRAM
Fast Page Mode
Asynchronous
SDRAM
Synchronous
EDO DRAM
Extended Data Output
Asynchronous
BEDO DRAM
Burst EDO
Asynchronous

8. The storage cell in a SRAM is
   (a) a flip-flop **(b)** a capacitor **(c)** a fuse **(d)** a magnetic domain
9. A DRAM must be
   (a) replaced periodically **(b)** refreshed periodically
   (c) always enabled **(d)** programmed before each use
10. A flash memory is
    (a) volatile **(b)** a read-only memory
    (c) a read/write memory **(d)** nonvolatile
    (e) answers (a) and (c) **(f)** answers (c) and (d)
11. Hard disk, floppy disk, Zip disk, and Jaz disk are all
    (a) magneto-optical storage devices **(b)** semiconductor storage devices
    (c) magnetic storage devices **(d)** optical storage devices
12. Optical storage devices employ
    (a) ultraviolet light **(b)** electromagnetic fields
    (c) optical couplers **(d)** lasers

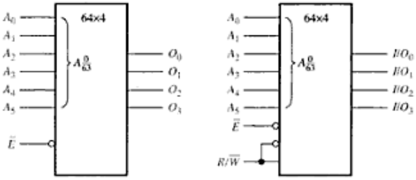
PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 10–1 Basics of Semiconductor Memory

1. Identify the ROM and the RAM in Figure 10–73.

FIGURE 10–73



(a) (b)

2. Explain why RAMs and ROMs are both random-access memories.
3. Explain the purposes of the address bus and the data bus.
4. What memory address (0 through 256) is represented by each of the following hexadecimal numbers:
   (a) $0A_{16}$ **(b)** $3F_{16}$ **(c)** CD_{16}

SECTION 10–2 Random-Access Memories (RAMs)

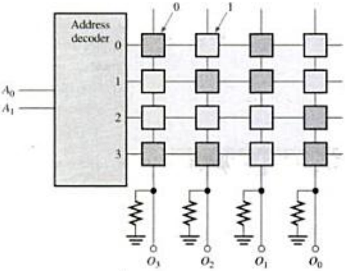
5. A static memory array with four rows similar to the one in Figure 10–8 is initially storing all 0s. What is its content after the following conditions? Assume a 1 selects a row.
   Row 0 = 1, Data in (Bit 0) = 1
   Row 1 = 0, Data in (Bit 1) = 1
   Row 2 = 1, Data in (Bit 2) = 1
   Row 3 = 0, Data in (Bit 3) = 0
6. Draw a basic logic diagram for a 512 × 8-bit static RAM, showing all the inputs and outputs.

7. Assuming that a 64k × 8 SRAM has a structure similar to that of the SRAM in Figure 10–10, determine the number of rows and 8-bit columns in its memory cell array.
8. Redraw the block diagram in Figure 10–10 for a 64k × 8 memory.
9. Explain the difference between a SRAM and a DRAM.
10. What is the capacity of a DRAM that has twelve address lines?

SECTION 10–3 Read-Only Memories (ROMs)

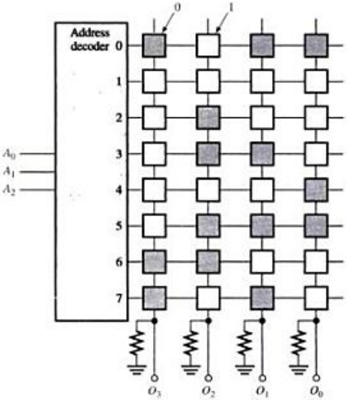
11. For the ROM array in Figure 10–74, determine the outputs for all possible input combinations, and summarize them in tabular form (Blue cell is a 1, gray cell is a 0).

FIGURE 10–74

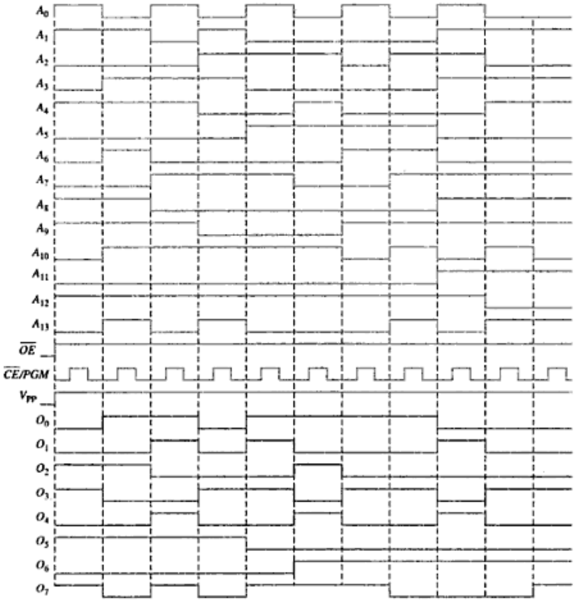


12. Determine the truth table for the ROM in Figure 10–75.

FIGURE 10–75

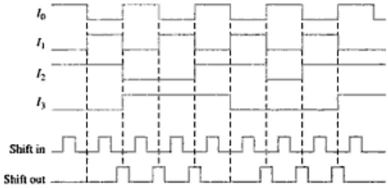


13. Using a procedure similar to that in Example 10–1, design a ROM for conversion of single-digit BCD to excess-3 code.
14. What is the total *bit* capacity of a ROM that has 14 address lines and 8 data outputs?


A_0
A_1
A_2
A_3
A_4
A_5
A_6
A_7
A_8
A_9
A_{10}
A_{11}
A_{12}
A_{13}
$\overline{OE}$
$\overline{CE}/PGM$
V_{PP}
O_0
O_1
O_2
O_3
O_4
O_5
O_6
O_7

▲ **FIGURE 10–77**

▶ **FIGURE 10–78**


I_0
I_1
I_2
I_3
Shift in
Shift out

22. In the memory of Problem 21, sixteen bytes are pushed into the stack. At what address is the first byte in located? At what address is the last byte in located?

SECTION 10–8 Magnetic and Optical Storage

23. Describe the general format of a hard disk.
24. Explain seek time and latency period in a hard disk drive.

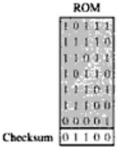
25. Why does magnetic tape require a much longer access time than does a disk?

26. Explain the differences in a magneto-optical disk, a CD-ROM, and a WORM.

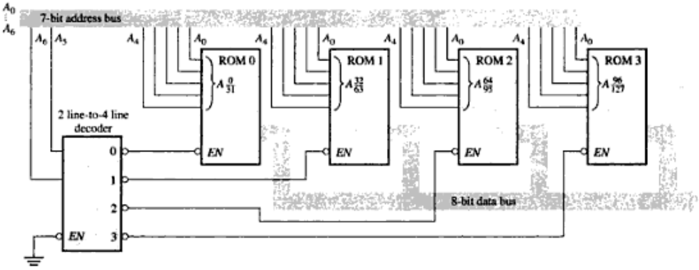
SECTION 10–9 **Testing Memory Chips**

27. Determine if the contents of the ROM in Figure 10–79 are correct.

► FIGURE 10–79


ROM
1 0 1 1 1
1 1 1 1 0
1 1 0 1 1
1 0 1 1 0
1 1 1 0 1
1 1 1 0 0
0 0 0 0 1
Checksum
0 1 1 0 0

28. A 128 × 8 ROM is implemented as shown in Figure 10–80. The decoder decodes the two most significant address bits to enable the ROMs one at a time, depending on the address selected.
    - **(a)** Express the lowest address and the highest address of each ROM as hexadecimal numbers.
    - **(b)** Assume that a single checksum is used for the entire memory and it is stored at the highest address. Develop a flowchart for testing the complete memory system.
    - **(c)** Assume that each ROM has a checksum stored at its highest address. Modify the flowchart developed in part (b) to accommodate this change.
    - **(d)** What is the disadvantage of using a single checksum for the entire memory rather than a checksum for each individual ROM?


A_0
A_6
7-bit address bus
A_6
A_5
A_4
A_0
ROM 0
A_{0}^{31}
ROM 1
A_{32}^{63}
ROM 2
A_{64}^{95}
ROM 3
A_{96}^{127}
2 line-to-4 line decoder
0
1
2
3
EN
8-bit data bus

▲ FIGURE 10–80

29. Suppose that a checksum test is run on the memory in Figure 10–80 and each individual ROM has a checksum at its highest address. What IC or ICs will you replace for each of the following error messages that appear on the system's video monitor?
    - **(a)** ADDRESSES 40–5F FAULTY
    - **(b)** ADDRESSES 20–3F FAULTY
    - **(c)** ADDRESSES 00–7F FAULTY

SECTION 10–8 Magnetic and Optical Storage

1. Magnetic storage: floppy disk, hard disk, tape, and magneto-optical disk.
2. Floppy disk storage capacity: 1.44 Mbytes
3. A magnetic disk is organized in tracks and sectors.
4. A magneto-optical disk uses a laser beam and an electromagnet.
5. Optical storage: CD-ROM, CD-R, CD-RW, DVD-ROM, WORM

SECTION 10–9 Testing Memory Chips

1. The contents of the ROM are added and compared with a prestored checksum.
2. Checksum cannot be used because the contents of a RAM are not fixed.
3. (1) a short between adjacent cells; (2) an inability of some cells to store both 1s and 0s; (3) dynamic altering of the contents of one address when the contents of another address change.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

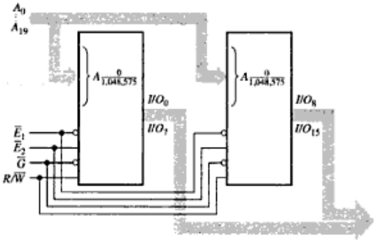
10–1 $G_3G_2G_1G_0 = 1110$

10–2 Connect eight 64k × 1 ROMs in parallel to form a 64k × 8 ROM.

10–3 Sixteen 64k × 1 ROMs

10–4 See Figure 10-81.

▶ FIGURE 10–81



10–5 ROM 1: 0 to 524,287; ROM 2: 524,288 to 1,048,575

SELF-TEST

1. (b) **2.** (c) **3.** (c) **4.** (d) **5.** (a) **6.** (d)
7. (c) **8.** (a) **9.** (b) **10.** (f) **11.** (c) **12.** (d)

11

INTEGRATED CIRCUIT TECHNOLOGIES

CHAPTER OBJECTIVES

- Identify fixed-function digital integrated circuits according to their complexity
- Make basic comparisons between the major IC technologies—CMOS and TTL
- Explain how the different series within the CMOS and TTL families differ from each other
- Define *propagation delay time, power dissipation, speed-power product,* and *fan-out* in relation to logic gates
- Determine the noise margin of a device from data sheet parameters
- Calculate the power dissipation of a device
- Explain how propagation delay affects the frequency of operation or speed of a circuit
- Interpret the speed-power product as a measure of performance
- Use data sheets to obtain information about a specific device
- Explain what the fan-out of a gate means
- Describe how basic TTL and CMOS gates operate at the component level
- Recognize the difference between TTL totem-pole outputs and TTL open-collector outputs and understand the limitations and uses of each
- Connect circuits in a wired-AND configuration
- Describe the operation of tristate circuits
- Properly terminate unused gate inputs
- Compare the performance of TTL and CMOS families
- Handle CMOS devices without risk of damage due to electrostatic discharge
- State the advantages of ECL
- Describe the PMOS and NMOS circuits
- Describe an E^2CMOS cell

INTRODUCTION

In section 1–4 and 3–7 you learned about the packages, pin numbers and examples of IC gates. This chapter provides the basics of digital integrated circuits such as complexity classification of digital ICs and the circuit technologies used to implement the gates as well as other types of IC devices.

Two major IC technologies, CMOS and TTL, are covered and their operating parameters are defined. Also, the operational characteristics of various families within these circuit technologies are compared. Other circuit technologies are also introduced. It is important to keep in mind that the particular circuit technology used to implement a logic gate has no effect on the logic operation of the gate. In terms of its truth table operation, a certain type of gate that is implemented with CMOS is the same as that type of gate implemented with TTL. The only differences in the gates are the electrical characteristics such as power dissipation, switching speed, and noise immunity.

All or portions of this chapter can be covered at any one of several appropriate points throughout the book or completely omitted, depending on the course objectives.

11–1 BASICS OF DIGITAL INTEGRATED CIRCUITS

Digital integrated circuits were introduced in Section 1–4 from the point of view of packages and pin numbers. Examples of IC gates were given in Section 3–7. Some of the basic concepts such as, IC classification based on circuit complexities and circuit technologies, most widely used IC technologies CMOS and TTL, performance parameters are introduced in this section. The IC technologies at the circuit component level will be covered in subsequent sections.

After completing this section, you should be able to

▪ Explain the complexity classification of ICs ▪ Identify the most common CMOS and TTL series ▪ Compare CMOS and TTL in terms of device types and performance parameters ▪ Define *propagation delay time* ▪ Define *power dissipation* ▪ Define *fan-out* ▪ Define *speed-power product* ▪ Interpret basic data sheet information

Complexity Classifications for Fixed-Function ICs

Fixed-function digital ICs are classified according to their complexity. They are listed here from the least complex to the most complex. The complexity figures stated here for SSI, MSI, LSI, VLSI, and ULSI are generally accepted, but definitions may vary from one source to another.

- **Small-scale integration (SSI)** describes fixed-function ICs that have up to twelve equivalent gate circuits on a single chip, and they include basic gates and flip-flops.
- **Medium-scale integration (MSI)** describes integrated circuits that have from 12 to 99 equivalent gates on a chip. They include logic functions such as encoders, decoders, counters, registers, multiplexers, arithmetic circuits, small memories, and others.
- **Large-scale integration (LSI)** is a classification of ICs with complexities of 100 to 9999 equivalent gates per chip, including memories.
- **Very large-scale integration (VLSI)** describes integrated circuits with complexities of 10,000 to 99,999 equivalent gates per chip.
- **Ultra large-scale integration (ULSI)** describes very large memories, larger **microprocessors**, and larger single-chip computers. Complexities of 100,000 equivalent gates and greater are classified as ULSI.

Integrated Circuit Technologies

The types of transistors with which all integrated circuits are implemented are either bipolar junction transistors or MOSFETs (metal-oxide semiconductor field-effect transistors). Two types of digital circuit technology that use bipolar junction transistors are TTL (transistor-transistor logic) and ECL (emitter-coupled logic). Of these two, TTL is more widely used. The major circuit technologies that use MOSFETs are CMOS (complementary MOS) and NMOS (*n*-channel MOS). Microprocessors use MOS technology.

All gates and other functions can be implemented with either type of circuit technology. SSI and MSI circuits are generally available in both TTL and CMOS. LSI, VLSI, and ULSI are generally implemented with CMOS or NMOS because it requires less area on a chip and consumes less power. Detailed discussion on these integrated technologies will be covered in this section and complete circuit-level coverage will be introduced in later sections. CMOS

TTL

TTL has been and still is a popular digital IC technology. One advantage of TTL is that it is not sensitive to electrostatic discharge as CMOS is and, therefore, is more practical in most laboratory experimentation and prototyping because you do not have to worry about handling precautions.

TTL Series Like CMOS, several series of TTL logic gates are available, all which operate from a 5 V dc supply. These series within the TTL family differ in their performance characteristics and are designated by the prefix 74 or 54 followed by a letter or letters that indicate the series and a number that indicates the type of logic device within the series. A TTL IC can be distinguished from CMOS by the letters that follow the 74 or 54 prefix.

The basic TTL series and their designations are as follows:

- 74—standard TTL (no letter)
- 74S—Schottky TTL
- 74AS—Advanced Schottky TTL
- 74LS—Low-power Schottky TTL
- 74ALS—Advanced Low-power Schottky TTL
- 74F—Fast TTL

Performance Characteristics and Parameters

Several things define the performance of a logic circuit. These performance characteristics are the switching speed measured in terms of the propagation delay time, the power dissipation, the fan-out or drive capability, the speed-power product, the dc supply voltage, and the input/output logic levels. We will define each of these and then compare the performance of CMOS and TTL.

Propagation Delay Time This parameter is a result of the limitation on switching speed or frequency at which a logic circuit can operate. The terms *low speed* and *high speed*, applied to logic circuits refer to the propagation delay time. The shorter the propagation delay, the higher the speed of the circuit and the higher the frequency at which it can operate.

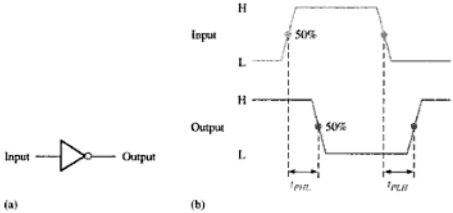
Propagation delay time, t_P, of a logic gate is the time interval between the application of an input pulse and the occurrence of the resulting output pulse. There are two different measurements of propagation delay time associated with a logic gate that apply to all the types of basic gates:

- t_{PHL}: The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from the HIGH level to the LOW level (HL).
- t_{PLH}: The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from the LOW level to the HIGH level (LH).

For standard-series TTL gates, the typical propagation delay is 11 ns and for F-series gates it is 3.3 ns. For HCT-series CMOS, the propagation delay is 7 ns, for the AC series it is 5 ns, and for the ALVC series it is 3 ns. All specified values are dependent on certain operating conditions as stated on a data sheet. For comparison, one type of ECL has a typical propagation delay of 0.22 ns.

EXAMPLE 11–1

Show the propagation delay times of the inverter in Figure 11–1(a).



▲ **FIGURE 11–1**

Solution The propagation delay times, t_{PHL} and t_{PLH}, are indicated in part (b) of the figure. In this case, the delays are measured between the 50% points of the corresponding edges of the input and output pulses. The values of t_{PHL} and t_{PLH} are not necessarily equal but in many cases they are the same.

Supplementary Problem One type of logic gate has a specified maximum t_{PLH} and t_{PHL} of 10 ns. For another type of gate the value is 4 ns. Which gate can operate at the highest frequency?

DC Supply Voltage (V_{CC}) The typical dc supply voltage for CMOS is either 5 V or 3.3 V, depending on the category. An advantage of CMOS is that the supply voltages can vary over a wider range than for TTL. The 5 V CMOS can tolerate supply variations from 2 V to 6 V and still operate properly although propagation delay time and power dissipation are significantly affected. The 3.3 V CMOS can operate with supply voltages from 2 V to 3.6 V. The typical dc supply voltage for TTL is 5.0 V with a minimum of 4.5 V and a maximum of 5.5 V.

Power Dissipation The **power dissipation**, P_D, of a logic gate is the product of the dc supply voltage and the average supply current. Normally, the supply current when the gate output is LOW is greater than when the gate output is HIGH. The manufacturer's data sheet usually designates the supply current for the LOW output state as I_{CCL} and for the HIGH state as I_{CCH}. The average supply current is determined based on a 50% duty cycle (output LOW half the time and HIGH half the time), so the average power dissipation of a logic gate is

Equation 11–1

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right)$$

CMOS series gates have very low power dissipations compared to the TTL series. However, the power dissipation of CMOS is dependent on the frequency of operation. At zero frequency the quiescent power is typically in the microwatt/gate range, and at the maximum operating frequency it can be in the low milliwatt range; therefore, power is sometimes specified at a given frequency. The HC series, for example, has a power of 2.75 μW/gate at 0 Hz (quiescent) and 600 μW/gate at 1 MHz.

Power dissipation for TTL is independent of frequency. For example, the ALS series uses 1.4 mW/gate regardless of the frequency and the F series uses 6 mW/gate.

Input and Output Logic Levels V_{IL} is the LOW level input voltage for a logic gate, and V_{IH} is the HIGH level input voltage. The 5 V CMOS accepts a maximum voltage of 1.5 V as V_{IL} and a minimum voltage of 3.5 V as V_{IH}. TTL accepts a maximum voltage of 0.8 V as V_{IL} and a minimum voltage of 2 V as V_{IH}.

V_{OL} is the LOW level output voltage and V_{OH} is the HIGH level output voltage. For 5 V CMOS, the maximum V_{OL} is 0.33 V and the minimum V_{OH} is 4.4 V. For TTL, the maximum V_{OL} is 0.4 V and the minimum V_{OH} is 2.4 V. All values depend on operating conditions as specified on the data sheet.

Speed-Power Product (SPP) This parameter (**speed-power product**) can be used as a measure of the performance of a logic circuit taking into account the propagation delay time and the power dissipation. It is especially useful for comparing the various logic gate series within the CMOS or TTL family or for comparing a CMOS gate to a TTL gate.

The SPP of a logic circuit is the product of the propagation delay time and the power dissipation and is expressed in joules (J), which is the unit of energy. The formula is

$$SPP = t_p P_D$$

Equation 11–2

EXAMPLE 11–2

A certain gate has a propagation delay of 5 ns and $I_{CCH} = 1$ mA and $I_{CCL} = 2.5$ mA with a dc supply voltage of 5 V. Determine the speed-power product.

Solution

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right) = 5\text{ V}\left(\frac{1\text{ mA} + 2.5\text{ mA}}{2}\right) = 5\text{ V}(1.75\text{ mA}) = 8.75\text{ mW}$$

$$SPP = (5\text{ ns})(8.75\text{ mW}) = \mathbf{43.75\ pJ}$$

Supplementary Problem

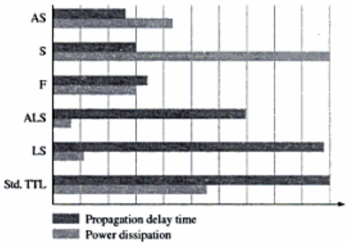
If the propagation delay of a gate is 15 ns and its *SPP* is 150 pJ, what is its average power dissipation?

Fan-Out and Loading The **fan-out** of a logic gate is the maximum number of inputs of the same series in an IC family that can be connected to a gate's output and still maintain the output voltage levels within specified limits. Fan-out is a significant parameter only for TTL because of the type of circuit technology. Since very high impedances are associated with CMOS circuits, the fan-out is very high but depends on frequency because of capacitive effects.

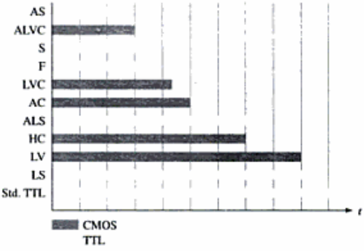
Fan-out is specified in terms of **unit loads**. A unit load for a logic gate equals one input to a like circuit. For example, a unit load for a 7400 NAND gate equals *one* input to another logic gate in the standard 74 series (not necessarily a NAND gate). Because, as specified in the 7400 data, the current to a HIGH input (I_{IH}) of a 7400 gate is 40 μA and the current from a LOW input (I_{IL}) is 1.6 mA, the number of unit loads that a 7400 gate can drive in both output states is

$$\text{Unit loads} = \frac{I_{OH}}{I_{IH}} = \frac{I_{OL}}{I_{IL}} = \frac{400\ \mu\text{A}}{40\ \mu\text{A}} = \frac{16\text{ mA}}{1.6\text{ mA}} = 10$$

Figure 11–2 shows LS logic gates driving a number of other gates of the same circuit technology, where the number of gates depends on the particular circuit technology. For example, as you have seen, the maximum number of gate inputs (unit loads) that a standard 74 series TTL gate can drive is 10. Most of the other TTL series, such as the LS, can drive 20 unit loads.



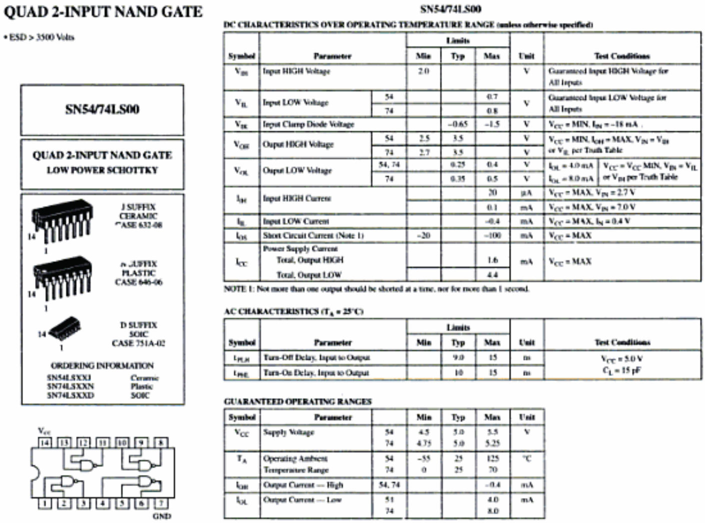
▲ **FIGURE 11–4**



▲ **FIGURE 11–5**

Data Sheets

A typical data sheet consists of an information page that shows, among other things, the logic diagram and packages, the recommended operating conditions, the electrical characteristics, and the switching characteristics. Partial data sheets for a 74LS00 and a 74HC00A are shown in Figures 11–6 and 11–7, respectively. The length of data sheets vary and some have much more information than others. Manufacturers' catalogues can be referred to find the data sheets of any device.



QUAD 2-INPUT NAND GATE

• ESD > 3500 Volts

SN54/74LS00

QUAD 2-INPUT NAND GATE
LOW POWER SCHOTTKY

14
1
J SUFFIX
CERAMIC
CASE 632-08
14
1
N SUFFIX
PLASTIC
CASE 646-06
14
1
D SUFFIX
SOIC
CASE 751A-02

ORDERING INFORMATION

SN54LSXXJ	Ceramic
SN74LSXXN	Plastic
SN74LSXXD	SOIC

V_{CC}
14
13
12
11
10
9
8
1
2
3
4
5
6
7
GND

SN54/74LS00

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

Symbol	Parameter		Limits Min	Typ	Max	Unit	Test Conditions
V_{IH}	Input HIGH Voltage		2.0			V	Guaranteed Input HIGH Voltage for All Inputs
V_{IL}	Input LOW Voltage	54			0.7	V	Guaranteed Input LOW Voltage for All Inputs
		74			0.8		
V_{IK}	Input Clamp Diode Voltage			−0.65	−1.5	V	V_{CC} = MIN, I_{IN} = −18 mA
V_{OH}	Output HIGH Voltage	54	2.5	3.5		V	V_{CC} = MIN, I_{OH} = MAX, V_{IN} = V_{IH} or V_{IL} per Truth Table
		74	2.7	3.5		V	
V_{OL}	Output LOW Voltage	54, 74		0.25	0.4	V	I_{OL} = 4.0 mA V_{CC} = V_{CC} MIN, V_{IN} = V_{IL} or V_{IH} per Truth Table
		74		0.35	0.5	V	I_{OL} = 8.0 mA
I_{IH}	Input HIGH Current				20	µA	V_{CC} = MAX, V_{IN} = 2.7 V
					0.1	mA	V_{CC} = MAX, V_{IN} = 7.0 V
I_{IL}	Input LOW Current				−0.4	mA	V_{CC} = MAX, I_{N} = 0.4 V
I_{OS}	Short Circuit Current (Note 1)		−20		−100	mA	V_{CC} = MAX
I_{CC}	Power Supply Current Total, Output HIGH				1.6	mA	V_{CC} = MAX
	Total, Output LOW				4.4		

NOTE 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (T_A = 25°C)

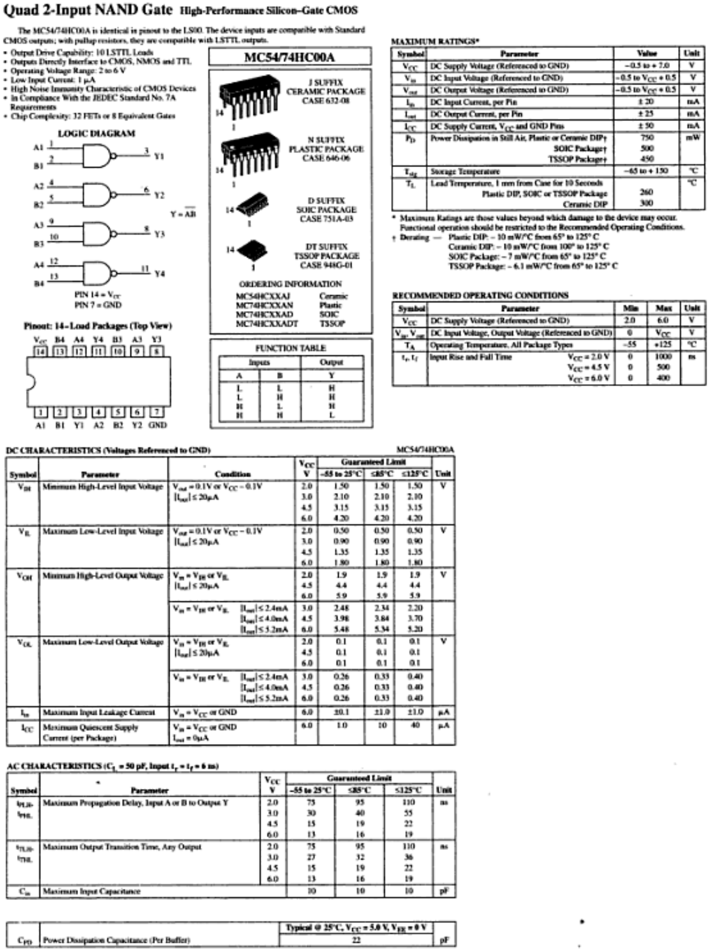
Symbol	Parameter	Limits Min	Typ	Max	Unit	Test Conditions
t_{PLH}	Turn-Off Delay, Input to Output		9.0	15	ns	V_{CC} = 5.0 V C_L = 15 pF
t_{PHL}	Turn-On Delay, Input to Output		10	15	ns	

GUARANTEED OPERATING RANGES

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54 74	4.5 4.75	5.0 5.0	5.5 5.25	V
T_A	Operating Ambient Temperature Range	54 74	−55 0	25 25	125 70	°C
I_{OH}	Output Current — High	54, 74			−0.4	mA
I_{OL}	Output Current — Low	54 74			4.0 8.0	mA

▲ **FIGURE 11–6**

The partial data sheet for a 74LS00



Quad 2-Input NAND Gate High-Performance Silicon–Gate CMOS

The MC54/74HC00A is identical in pinout to the LS00. The device inputs are compatible with Standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS and TTL
- Operating Voltage Range: 2 to 6 V
- Low Input Current: 1 μA
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance With the JEDEC Standard No. 7A Requirements
- Chip Complexity: 32 FETs or 8 Equivalent Gates

MC54/74HC00A

J SUFFIX
CERAMIC PACKAGE
CASE 632-08

N SUFFIX
PLASTIC PACKAGE
CASE 646-06

D SUFFIX
SOIC PACKAGE
CASE 751A-03

DT SUFFIX
TSSOP PACKAGE
CASE 948G-01

ORDERING INFORMATION

MC54HCXXAJ	Ceramic
MC74HCXXAN	Plastic
MC74HCXXAD	SOIC
MC74HCXXADT	TSSOP

LOGIC DIAGRAM

PIN 14 = V_{CC}
PIN 7 = GND

Pinout: 14–Load Packages (Top View)

FUNCTION TABLE

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

MAXIMUM RATINGS*

Symbol	Parameter	Value	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	–0.5 to + 7.0	V
V_{in}	DC Input Voltage (Referenced to GND)	–0.5 to V_{CC} + 0.5	V
V_{out}	DC Output Voltage (Referenced to GND)	–0.5 to V_{CC} + 0.5	V
I_{in}	DC Input Current, per Pin	± 20	mA
I_{out}	DC Output Current, per Pin	± 25	mA
I_{CC}	DC Supply Current, V_{CC} and GND Pins	± 50	mA
P_D	Power Dissipation in Still Air, Plastic or Ceramic DIP†	750	mW
	SOIC Package†	500	
	TSSOP Package†	450	
T_{stg}	Storage Temperature	–65 to + 150	°C
T_L	Lead Temperature, 1 mm from Case for 10 Seconds		°C
	Plastic DIP, SOIC or TSSOP Package	260	
	Ceramic DIP	300	

* Maximum Ratings are those values beyond which damage to the device may occur. Functional operation should be restricted to the Recommended Operating Conditions.

† Derating — Plastic DIP: – 10 mW/°C from 65° to 125° C
Ceramic DIP: – 10 mW/°C from 100° to 125° C
SOIC Package: – 7 mW/°C from 65° to 125° C
TSSOP Package: – 6.1 mW/°C from 65° to 125° C

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	2.0	6.0	V
V_{in}, V_{out}	DC Input Voltage, Output Voltage (Referenced to GND)	0	V_{CC}	V
T_A	Operating Temperature, All Package Types	–55	+125	°C
t_r, t_f	Input Rise and Fall Time V_{CC} = 2.0 V	0	1000	ns
	V_{CC} = 4.5 V	0	500	
	V_{CC} = 6.0 V	0	400	

DC CHARACTERISTICS (Voltages Referenced to GND) MC54/74HC00A

Symbol	Parameter	Condition	V_{CC} V	Guaranteed Limit –55 to 25°C	≤85°C	≤125°C	Unit
V_{IH}	Minimum High-Level Input Voltage	V_{out} = 0.1V or V_{CC} – 0.1V $\lvert I_{out}\rvert \le 20\mu A$	2.0	1.50	1.50	1.50	V
			3.0	2.10	2.10	2.10	
			4.5	3.15	3.15	3.15	
			6.0	4.20	4.20	4.20	
V_{IL}	Maximum Low-Level Input Voltage	V_{out} = 0.1V or V_{CC} – 0.1V $\lvert I_{out}\rvert \le 20\mu A$	2.0	0.50	0.50	0.50	V
			3.0	0.90	0.90	0.90	
			4.5	1.35	1.35	1.35	
			6.0	1.80	1.80	1.80	
V_{OH}	Minimum High-Level Output Voltage	V_{in} = V_{IH} or V_{IL} $\lvert I_{out}\rvert \le 20\mu A$	2.0	1.9	1.9	1.9	V
			4.5	4.4	4.4	4.4	
			6.0	5.9	5.9	5.9	
		V_{in} = V_{IH} or V_{IL} $\lvert I_{out}\rvert \le 2.4mA$	3.0	2.48	2.34	2.20	
		$\lvert I_{out}\rvert \le 4.0mA$	4.5	3.98	3.84	3.70	
		$\lvert I_{out}\rvert \le 5.2mA$	6.0	5.48	5.34	5.20	
V_{OL}	Maximum Low-Level Output Voltage	V_{in} = V_{IH} or V_{IL} $\lvert I_{out}\rvert \le 20\mu A$	2.0	0.1	0.1	0.1	V
			4.5	0.1	0.1	0.1	
			6.0	0.1	0.1	0.1	
		V_{in} = V_{IH} or V_{IL} $\lvert I_{out}\rvert \le 2.4mA$	3.0	0.26	0.33	0.40	
		$\lvert I_{out}\rvert \le 4.0mA$	4.5	0.26	0.33	0.40	
		$\lvert I_{out}\rvert \le 5.2mA$	6.0	0.26	0.33	0.40	
I_{in}	Maximum Input Leakage Current	V_{in} = V_{CC} or GND	6.0	±0.1	±1.0	±1.0	μA
I_{CC}	Maximum Quiescent Supply Current (per Package)	V_{in} = V_{CC} or GND I_{out} = 0μA	6.0	1.0	10	40	μA

AC CHARACTERISTICS (C_L = 50 pF, Input t_r = t_f = 6 ns)

Symbol	Parameter	V_{CC} V	Guaranteed Limit –55 to 25°C	≤85°C	≤125°C	Unit
t_{PLH}, t_{PHL}	Maximum Propagation Delay, Input A or B to Output Y	2.0	75	95	110	ns
		3.0	30	40	55	
		4.5	15	19	22	
		6.0	13	16	19	
t_{TLH}, t_{THL}	Maximum Output Transition Time, Any Output	2.0	75	95	110	ns
		3.0	27	32	36	
		4.5	15	19	22	
		6.0	13	16	19	
C_{in}	Maximum Input Capacitance		10	10	10	pF

Symbol	Parameter	Typical @ 25°C, V_{CC} = 5.0 V, V_{EE} = 0 V	Unit
C_{PD}	Power Dissipation Capacitance (Per Buffer)	22	pF

▲ **FIGURE 11–7**

The partial data sheet for a 74HC00A

SECTION 11–1 REVIEW

1. Define the terms SSI, MSI, LSI, VLSI, and ULSI.
2. Generally, in what classification does a fixed-function IC with the following number of equivalent gates fall?
   (a) 75 (b) 500 (c) 10 (d) 15,000 (e) 200,000
3. List three types of IC technologies and name the two that are the most widely used.
4. Identify the following IC logic designators:
   (a) LS (b) ALS (c) F (d) HC (e) AC (f) HCT (g) LV
5. Identify the following devices according to logic function:
   (a) 74LS04 (b) 74HC00 (c) 74LV08 (d) 74ALS10
   (a) 7432 (f) 74ACT11 (g) 74AHC02
6. Which IC technology generally has the lowest power dissipation?
7. What does the term *hex inverter* mean? What does *quad 2-input NAND* mean?
8. A positive pulse is applied to an inverter input. The time from the leading edge of the input to the leading edge of the output is 10 ns. The time from the trailing edge of the input to the trailing edge of the output is 8 ns. What are the values of t_{PLH} and t_{PHL}?
9. A certain gate has a propagation delay time of 6 ns and a power dissipation of 3 mW. Determine the speed-power product?
10. Define I_{CCL} and I_{CCH}.
11. Define V_{IL} and V_{IH}.
12. Define V_{OL} and V_{OH}.

11–2 BASIC OPERATIONAL CHARACTERISTICS AND PARAMETERS

When you work with digital ICs, you should be familiar, not only with their logical operation, but also with such operational properties as voltage levels, noise immunity, power dissipation, fan-out, and propagation delays. In this section, the practical aspects of these properties are discussed.

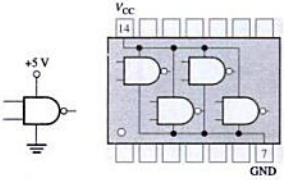
After completing this section, you should be able to

- Determine the power and ground connections
- Describe the logic levels for CMOS and TTL
- Discuss noise immunity
- Determine the power dissipation of a logic circuit
- Define the propagation delays of a logic gate
- Discuss speed-power product and explain its significance
- Discuss loading and fan-out of TTL and CMOS

DC Supply Voltage

The nominal value of the dc supply voltage for **TTL** (transistor-transistor logic) devices is +5 V. TTL is also designated T^2L. **CMOS** (complementary metal-oxide semiconductor) devices are available in two different supply voltage categories, +5 V and +3.3 V, the latter is known as low-voltage CMOS. Although omitted from logic diagrams for simplicity, the dc supply voltage is connected to the V_{CC} pin of an IC package, and ground is connected to the GND pin. Both voltage and ground are distributed internally to all elements within the package, as illustrated in Figure 11–8 for a 14-pin package.

FIGURE 11–8


+5 V
V_{CC}
14
7
GND

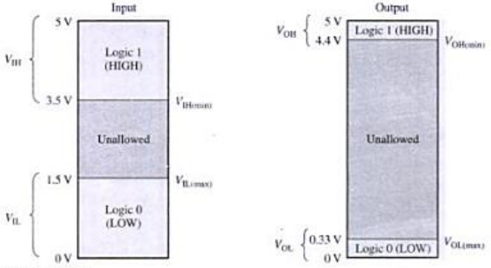
(a) Single gate (b) IC dual in-line package

CMOS Logic Levels

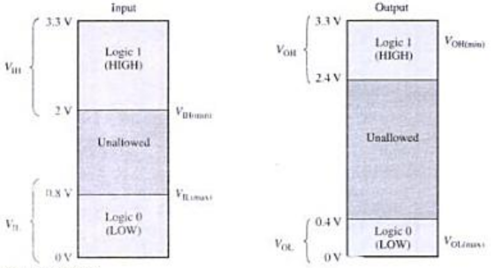
Logic levels were discussed briefly in Chapter 1. There are four different logic-level specifications: V_{IL}, V_{IH}, V_{OL}, and V_{OH}. For CMOS circuits, the ranges of input voltages (V_{IL}) that can represent a valid LOW (logic 0) are from 0 V to 1.5 V for the +5 V logic and 0 V to 0.8 V for the 3.3 V logic. The ranges of input voltages (V_{IH}) that can represent a valid HIGH (logic 1) are from 3.5 V to 5 V for the 5 V logic and 2 V to 3.3 V for the 3.3 V logic, as indicated in Figure 11–9. The ranges of values from 1.5 V to 3.5 V for 5 V logic and 0.8 V to 2 V for 3.3 V

FIGURE 11–9

Input and output logic levels for CMOS


Input
5 V
V_{IH}
Logic 1 (HIGH)
3.5 V
$V_{IH(min)}$
Unallowed
1.5 V
$V_{IL(max)}$
V_{IL}
Logic 0 (LOW)
0 V
Output
V_{OH}
5 V
4.4 V
Logic 1 (HIGH)
$V_{OH(min)}$
Unallowed
V_{OL}
0.33 V
0 V
Logic 0 (LOW)
$V_{OL(max)}$

(a) +5 V CMOS


Input
3.3 V
V_{IH}
Logic 1 (HIGH)
2 V
$V_{IH(min)}$
Unallowed
0.8 V
$V_{IL(max)}$
V_{IL}
Logic 0 (LOW)
0 V
Output
3.3 V
V_{OH}
Logic 1 (HIGH)
$V_{OH(min)}$
2.4 V
Unallowed
0.4 V
V_{OL}
Logic 0 (LOW)
$V_{OL(max)}$
0 V

(b) +3.3 V CMOS

logic are regions of unpredictable performance, and values in these ranges are disallowed. When an input voltage is in one of these ranges, it can be interpreted as either a HIGH or a LOW by the logic circuit. Therefore, CMOS gates cannot be operated reliably when the input voltages are in these unallowed ranges.

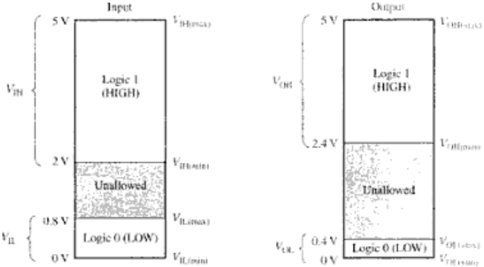
The ranges of CMOS output voltages (V_{OL} and V_{OH}) for both 5 V and 3.3 V logic are also shown in Figure 11–9. Notice that the minimum HIGH output voltage, $V_{OH(min)}$, is greater than the minimum HIGH input voltage, $V_{IH(min)}$. Also, notice that the maximum LOW output voltage, $V_{OL(max)}$, is less than the maximum LOW input voltage, $V_{IL(max)}$.

TTL Logic Levels

The input and output logic levels for TTL are given in Figure 11–10. Just as for CMOS, there are four different logic level specifications: V_{IL}, V_{IH}, V_{OL}, and V_{OH}.

FIGURE 11–10

Input and output logic levels for TTL



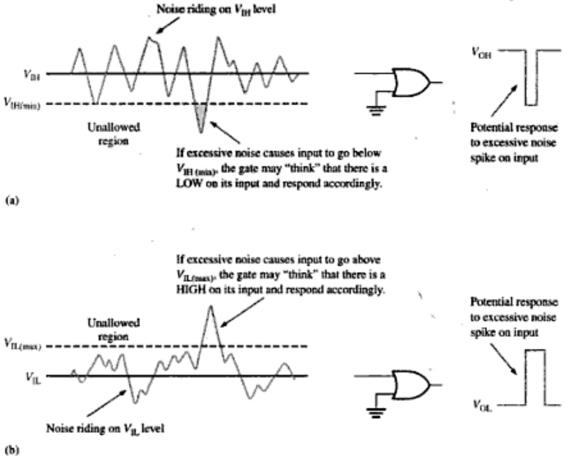
Noise Immunity

Noise is unwanted voltage that is induced in electrical circuits and can present a threat to the proper operation of the circuit. Wires and other conductors within a system can pick up stray high-frequency electromagnetic radiation from adjacent conductors in which currents are changing rapidly or from many other sources external to the system. Also, power-line voltage fluctuation is a form of low-frequency noise.

In order not to be adversely affected by noise, a logic circuit must have a certain amount of **noise immunity**. This is the ability to tolerate a certain amount of unwanted voltage fluctuation on its inputs without changing its output state. For example, if noise voltage causes the input of a 5 V CMOS gate to drop below 3.5 V in the HIGH state, the input is in the unallowed region and operation is unpredictable (see Figure 11–9). Thus, the gate may interpret the fluctuation below 3.5 V as a LOW level, as illustrated in Figure 11–11(a). Similarly, if noise causes a gate input to go above 1.5 V in the LOW state, an uncertain condition is created, as illustrated in part (b).

FIGURE 11–11

Illustration of the effects of input noise on gate operation


Noise riding on V_{IH} level
V_{IH}
$V_{IH(min)}$
Unallowed region
V_{OH}
(a)
Unallowed region
$V_{IL(max)}$
V_{IL}
Noise riding on V_{IL} level
V_{OL}
(b)

If excessive noise causes input to go below $V_{IH(min)}$, the gate may "think" that there is a LOW on its input and respond accordingly.

Potential response to excessive noise spike on input

If excessive noise causes input to go above $V_{IL(max)}$, the gate may "think" that there is a HIGH on its input and respond accordingly.

Potential response to excessive noise spike on input

Noise Margin

A measure of a circuit's noise immunity is called the **noise margin**, which is expressed in volts. There are two values of noise margin specified for a given logic circuit: the HIGH-level noise margin (V_{NH}) and the LOW-level noise margin (V_{NL}). These parameters are defined by the following equations:

Equation 11–3

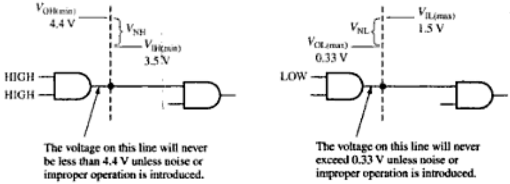
$$V_{NH} = V_{OH(min)} - V_{IH(min)}$$

Equation 11–4

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

Sometimes, you will see the noise margin expressed as a percentage of V_{CC}. From the equations, V_{NH} is the difference between the lowest possible HIGH output from a driving gate ($V_{OH(min)}$) and the lowest possible HIGH input that the load gate can tolerate ($V_{IH(min)}$). Noise margin V_{NL} is the difference between the maximum possible LOW input that a gate can tolerate ($V_{IL(max)}$) and the maximum possible LOW output of the driving gate ($V_{OL(max)}$). Noise margins are illustrated in Figure 11–12.

FIGURE 11–12


$V_{OH(min)}$
4.4 V
V_{NH}
$V_{IH(min)}$
3.5 V
HIGH
HIGH
$V_{IL(max)}$
1.5 V
V_{NL}
$V_{OL(max)}$
0.33 V
LOW

The voltage on this line will never be less than 4.4 V unless noise or improper operation is introduced.

The voltage on this line will never exceed 0.33 V unless noise or improper operation is introduced.

(a) HIGH-level noise margin

(b) LOW-level noise margin

cycle is 50%, the output is HIGH half the time and LOW the other half. The average supply current is therefore

Equation 11–5

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2}$$

The average power dissipation is

Equation 11–6

$$P_D = V_{CC}I_{CC}$$

EXAMPLE 11–4

A certain gate draws 2 μA when its output is HIGH and 3.6 μA when its output is LOW. What is its average power dissipation if V_{CC} is 5 V and the gate is operated on a 50% duty cycle?

Solution The average I_{CC} is

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2} = \frac{2.0\ \mu A + 3.6\ \mu A}{2} = 2.8\ \mu A$$

The average power dissipation is

$$P_D = V_{CC}I_{CC} = (5\ V)(2.8\ \mu A) = \mathbf{14\ \mu W}$$

Supplementary Problem A certain IC gate has an $I_{CCH} = 1.5\ \mu A$ and $I_{CCL} = 2.8\ \mu A$. Determine the average power dissipation for 50% duty cycle operation if V_{CC} is 5 V.

Power dissipation in a TTL circuit is essentially constant over its range of operating frequencies. Power dissipation in CMOS, however, is frequency dependent. It is extremely low under static (dc) conditions and increases as the frequency increases. These characteristics are shown in the general curves of Figure 11–14. For example, the power dissipation of a low-power Schottky (LS) TTL gate is a constant 2.2 mW. The power dissipation of an HCMOS gate is 2.75 μW under static conditions and 170 μW at 100 kHz.

Propagation Delay

When a signal passes (propagates) through a logic circuit, it always experiences a time delay, as illustrated in Figure 11–15. A change in the output level always occurs a short time, called the **propagation delay time**, later than the change in the input level that caused it.

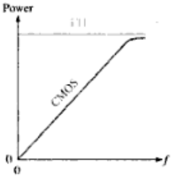


FIGURE 11–14

Power versus frequency curves for TTL and CMOS

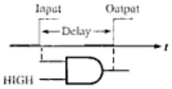


FIGURE 11–15

A basic illustration of propagation delay

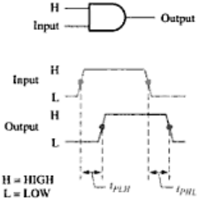
As mentioned in Section 11.1, there are two propagation delay times specified for logic gates:

- t_{PHL}: The time between a designated point on the input pulse and the corresponding point on the output pulse when the output is changing from HIGH to LOW.
- t_{PLH}: The time between a designated point on the input pulse and the corresponding point on the output pulse when the output is changing from LOW to HIGH.

These propagation delay times are illustrated in Figure 11–16, with the 50% points on the pulse edges used as references.

FIGURE 11–16

Propagation delay times



The propagation delay of a gate limits the frequency at which it can be operated. The greater the propagation delay, the lower the maximum frequency. Thus, a higher-speed circuit is one that has a smaller propagation delay. For example, a gate with a delay of 3 ns is faster than one with a 10 ns delay.

Speed-Power Product

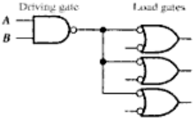
The speed-power product provides a basis for the comparison of logic circuits when *both* propagation delay and power dissipation are important considerations in the selection of the type of logic to be used in a certain application. The lower the speed-power product, the better. The unit of speed-power product is the picojoule (pJ). For example, HCMOS has a speed-power product of 1.2 pJ at 100 kHz while LS TTL has a value of 22 pJ.

Loading and Fan-Out

When the output of a logic gate is connected to one or more inputs of other gates, a load on the driving gate is created, as shown in Figure 11–17. There is a limit to the number of load gate inputs that a given gate can drive. This limit is called the **fan-out** of the gate.

FIGURE 11–17

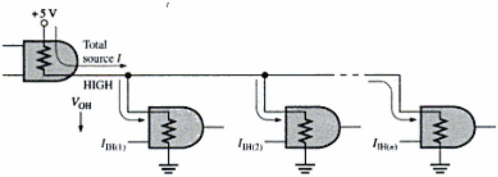
Loading a gate output with gate inputs



connected, V_{OH} drops below $V_{OH(min)}$, and the HIGH-level noise margin is reduced, thus compromising the circuit operation. Also, as the total source current increases, the power dissipation of the driving gate increases.

▶ **FIGURE 11–20**

HIGH-state TTL loading

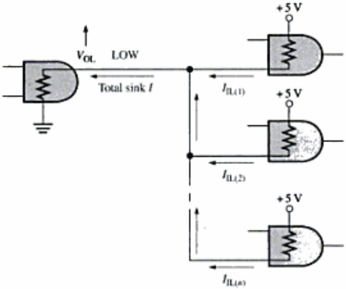


The fan-out is the maximum number of load gate inputs that can be connected without adversely affecting the specified operational characteristics of the gate. For example, low-power Schottky (LS) TTL has a fan-out of 20 unit loads. One input of the same logic family as the driving is called a **unit load.**

The total sink current also increases with each load gate input that is added, as shown in Figure 11–21. As this current increases, the internal voltage drop of the driving gate increases, causing V_{OL} to increase. If an excessive number of loads are added, V_{OL} exceeds $V_{OL(max)}$, and the LOW-level noise margin is reduced.

▶ **FIGURE 11–21**

LOW-stage TTL loading



In TTL, the current-sinking capability (LOW output state) is the limiting factor in determining the fan-out.

SECTION 11–2 REVIEW

Answers are at the end of the chapter.

1. Define V_{IH}, V_{IL}, V_{OH}, and V_{OL}.
2. Is it better to have a lower value of noise margin or a higher value?
3. Gate *A* has a greater propagation delay time than gate *B*. Which gate can operate at a higher frequency?
4. How does excessive loading affect the noise margin of a gate?

11–3 CMOS CIRCUITS

Basic internal CMOS circuitry and its operation are discussed in this section. The abbreviation CMOS stands for complementary metal-oxide semiconductor. The term *complementary* refers to the use of two types of transistors in the output circuit. An *n*-channel MOSFET (MOS field-effect transistor) and a *p*-channel MOSFET are used.

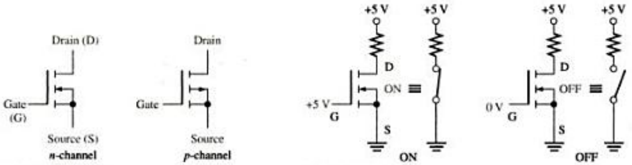
After completing this section, you should be able to

■ Identify a MOSFET by its symbol ■ Discuss the switching action of a MOSFET ■ Describe the basic operation of a CMOS inverter circuit ■ Describe the basic operation of CMOS NAND and NOR gates ■ Explain the operation of a CMOS gate with an open-drain output ■ Discuss the operation of tristate CMOS gates ■ List the precautions required when handling CMOS devices

The MOSFET

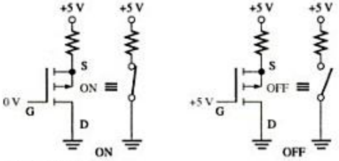
Metal-oxide semiconductor field-effect transistors (MOSFETs) are the active switching elements in CMOS circuits. These devices differ greatly in construction and internal operation from bipolar junction transistors used in TTL circuits, but the switching action is basically the same: they function ideally as open or closed switches, depending on the input.

Figure 11–22(a) shows the symbols for both *n*-channel and *p*-channel MOSFETs. As indicated, the three terminals of a MOSFET are **gate, drain,** and **source**. When the gate voltage of an *n*-channel MOSFET is more positive than the source, the MOSFET is on (*saturation*), and there is, ideally, a closed switch between the drain and the source. When the gate-to-source voltage is zero, the MOSFET is off (*cutoff*), and there is, ideally, an open switch between the drain and the source. This operation is illustrated in Figure 11–22(b). The *p*-channel MOSFET operates with opposite voltage polarities, as shown in part (c).



(a) MOSFET symbols

(b) *n*-channel switch



(c) *p*-channel switch

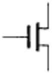
▲ **FIGURE 11–22**

Basic symbols and switching action of MOSFETs

Sometimes a simplified MOSFET symbol as shown in Figure 11–23 is used.

FIGURE 11–23

Simplified MOSFET symbol

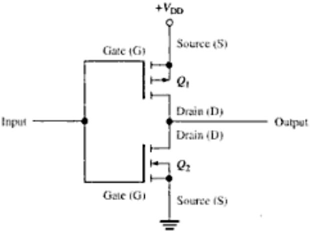


CMOS Inverter

Complementary MOS (CMOS) logic uses the MOSFET in complementary pairs as its basic element. A complementary pair uses both *p*-channel and *n*-channel enhancement MOSFETs, as shown in the inverter circuit in Figure 11–24.

FIGURE 11–24

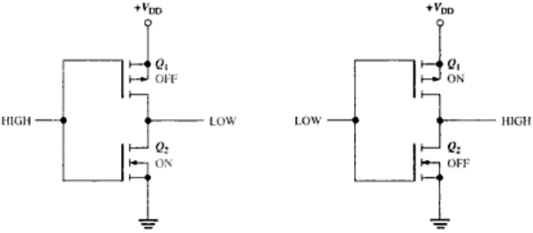
A CMOS inverter circuit



When a HIGH is applied to the input, as shown in Figure 11–25(a), the *p*-channel MOSFET Q_1 is off and the *n*-channel MOSFET Q_2 is on. This condition connects the output to ground through the *on* resistance of Q_2, resulting in a LOW output. When a LOW is applied to the input, as shown in Figure 11–25(b), Q_1 is on and Q_2 is off. This condition connects the output to $+V_{DD}$ (dc supply voltage) through the *on* resistance of Q_1, resulting in a HIGH output.

FIGURE 11–25

Operation of a CMOS inverter



(a) HIGH input, LOW output

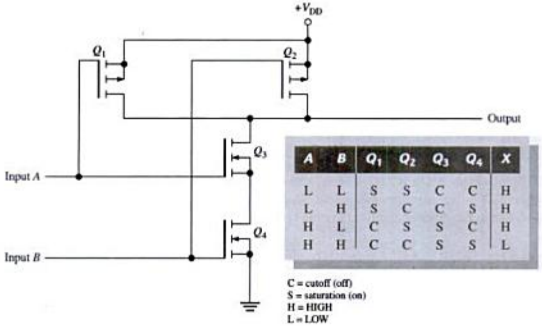
(b) LOW input, HIGH output

CMOS NAND Gate

Figure 11–26 shows a CMOS NAND gate with two inputs. Notice the arrangement of the complementary pairs (*n*-channel and *p*-channel MOSFETs).

▶ FIGURE 11–26

A CMOS NAND gate circuit



A	B	Q_1	Q_2	Q_3	Q_4	X
L	L	S	S	C	C	H
L	H	S	C	C	S	H
H	L	C	S	S	C	H
H	H	C	C	S	S	L

C = cutoff (off)
S = saturation (on)
H = HIGH
L = LOW

The operation of a CMOS NAND gate is follows:

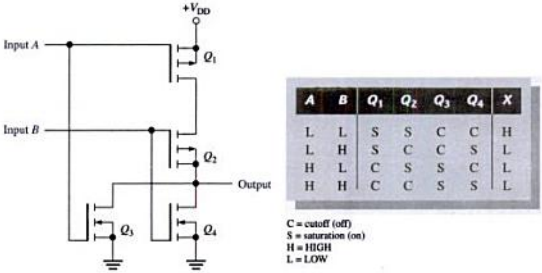
- When both inputs are LOW, Q_1 and Q_2 are on, and Q_3 and Q_4 are off. The output is pulled HIGH through the *on* resistance of Q_1 and Q_2 in parallel.
- When input A is LOW and input B is HIGH, Q_1 and Q_4 are on, and Q_2 and Q_3 are off. The output is pulled HIGH through the low *on* resistance of Q_1.
- When input A is HIGH and input B is LOW, Q_1 and Q_4 are off, and Q_2 and Q_3 are on. The output is pulled HIGH through the low *on* resistance of Q_2.
- Finally, when both inputs are HIGH, Q_1 and Q_2 are off, and Q_3 and Q_4 are on. In this case, the output is pulled LOW through the *on* resistance of Q_3 and Q_4 in series to ground.

CMOS NOR Gate

Figure 11–27 shows a CMOS NOR gate with two inputs. Notice the arrangement of the complementary pairs.

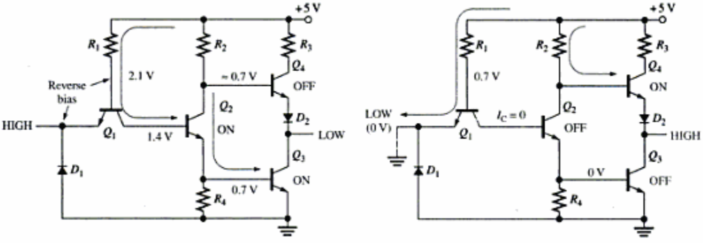
▶ FIGURE 11–27

A CMOS NOR gate circuit



A	B	Q_1	Q_2	Q_3	Q_4	X
L	L	S	S	C	C	H
L	H	S	C	C	S	L
H	L	C	S	S	C	L
H	H	C	C	S	S	L

C = cutoff (off)
S = saturation (on)
H = HIGH
L = LOW


+5 V
R_1
R_2
R_3
Q_4
OFF
2.1 V
≈ 0.7 V
Reverse bias
HIGH
Q_1
1.4 V
Q_2
ON
D_2
LOW
Q_3
ON
D_1
0.7 V
R_4
+5 V
R_1
R_2
R_3
Q_4
ON
0.7 V
LOW
(0 V)
$I_C = 0$
Q_1
Q_2
OFF
D_2
HIGH
Q_3
0 V
OFF
D_1
R_4

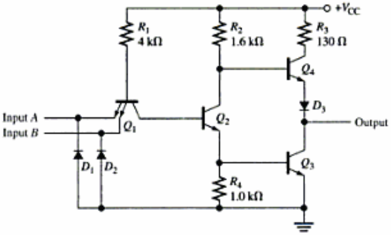
(a) (b)

▲ FIGURE 11-35

Operation of a TTL inverter

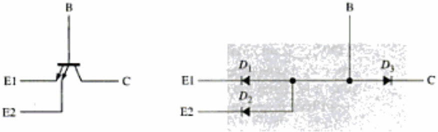
► FIGURE 11-36

A TTL NAND gate circuit


$+V_{CC}$
R_1 4 kΩ
R_2 1.6 kΩ
R_3 130 Ω
Q_4
D_3
Input A
Input B
Q_1
Q_2
Output
D_1
D_2
Q_3
R_4 1.0 kΩ

► FIGURE 11-37

Diode equivalent of a TTL multiple-emitter transistor


B
E1
C
E2
B
E1
D_1
D_3
C
D_2
E2

input B forward-biases the respective diode and reverse-biases D_3 (Q_1 base-collector junction). This action keeps Q_2 off and results in a HIGH output in the same way as described for the TTL inverter. Of course, a LOW on both inputs will do the same thing.

A HIGH on both inputs reverse-biases both input diodes and forward-biases D_3 (Q_1 base-collector junction). This action turns Q_2 on and results in a LOW output in the same way as described for the TTL inverter. You should recognize this operation as that of the NAND function: The output is LOW only if all inputs are HIGH.

Open-Collector Gates

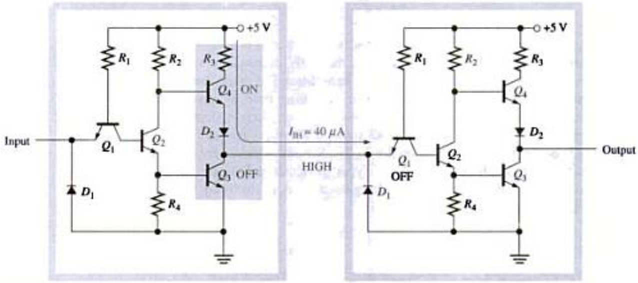
The TTL gates described in the previous sections all had the totem-pole output circuit. Another type of output available in TTL integrated circuits is the **open-collector** output. This

11–5 PRACTICAL CONSIDERATIONS IN THE USE OF TTL

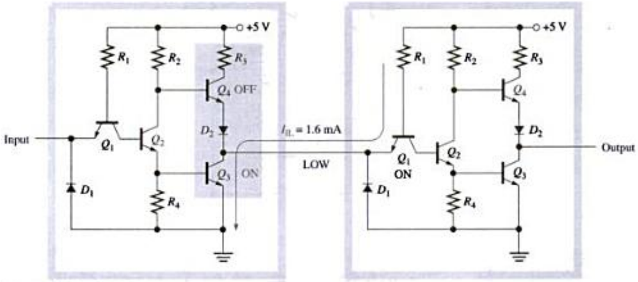
Although CMOS is the more predominant IC technology in industry and commercial applications, TTL is still used. In educational applications, TTL is usually preferred because it does not have the handling restrictions that CMOS does due to ESD. Because of this, several practical considerations in the use and application of TTL circuits will be covered using standard TTL for illustration.

After completing this section, you should be able to

- Describe current sinking and current sourcing
- Use an open-collector circuit for wired-AND operation
- Describe the effects of connecting two or more totem-pole outputs
- Use open-collector gates to drive LEDs and lamps
- Explain what to do with unused TTL inputs


+5 V
+5 V
R_1
R_2
R_3
Q_4
ON
D_2
I_{IH} = 40 μA
Input
Q_1
Q_2
Q_3
OFF
HIGH
D_1
R_4
R_1
R_2
R_3
Q_4
D_2
Output
Q_1
OFF
Q_2
Q_3
D_1
R_4

(a) Current sourcing (I_{IH} value is maximum)


+5 V
+5 V
R_1
R_2
R_3
Q_4
OFF
D_2
I_{IL} = 1.6 mA
Input
Q_1
Q_2
Q_3
ON
LOW
D_1
R_4
R_1
R_2
R_3
Q_4
D_2
Output
Q_1
ON
Q_2
Q_3
D_1
R_4

(b) Current sinking (I_{IL} value is maximum)

▲ FIGURE 11–43

Current sinking and sourcing action in TTL

Current Sinking and Current Sourcing

The concepts of current sinking and current sourcing were introduced in Section 11–2. Now that you are familiar with the totem-pole-output circuit configuration used in TTL, let us look closer at the sinking and sourcing action.

Figure 11–43 shows a standard TTL inverter with a totem-pole output connected to the input of another TTL inverter. When the driving gate is in the HIGH output state, the driver is sourcing current to the load, as shown in Figure 11–43(a). The input to the load gate is like a reverse-biased diode, so there is practically no current required by the load. Actually, since the input is nonideal, there is a maximum of 40 μA from the totem-pole output of the driver into the load gate input.

When the driving gate is in the LOW output state, the driver is sinking current from the load, as shown in Figure 11–43(b). This current is 1.6 mA maximum for standard TTL and is indicated on a **data sheet** with a negative value because it is *out* of the input.

EXAMPLE 11–5

When a TTL NAND gate drives five TTL inputs, how much current does the driver output source, and how much does it sink?

Solution Total source current (in HIGH output state):

$$I_{\text{IH(max)}} = 40\ \mu\text{A per input}$$

$$I_{\text{T(source)}} = (5\text{ inputs})(40\ \mu\text{A/input}) = 5(40\ \mu\text{A}) = \mathbf{200\ \mu A}$$

Total sink current (in LOW output state):

$$I_{\text{IL(max)}} = -1.6\text{ mA per input}$$

$$I_{\text{T(sink)}} = (5\text{ inputs})(-1.6\text{ mA/input}) = 5(-1.6\text{ mA}) = \mathbf{-8.0\ mA}$$

Supplementary Problem Repeat the calculations for an LS TTL gate. Refer to a data sheet.

EXAMPLE 11–6

Refer to the data sheet in the catalogue and determine the fan-out of the 7400 NAND gate.

Solution According to the data sheet, the current parameters are as follows:

$$I_{\text{IH(max)}} = 40\ \mu\text{A} \qquad I_{\text{OH(max)}} = -400\ \mu\text{A}$$

$$I_{\text{IL(max)}} = -1.6\text{ mA} \qquad I_{\text{OL(max)}} = 16\text{ mA}$$

Fan-out for the HIGH output state is calculated as follows: Current $I_{\text{OH(max)}}$ is the maximum current that the gate can source to a load. Each load input requires an $I_{\text{IH(max)}}$ of 40 μA. The HIGH-state fan-out is

$$\left|\frac{I_{\text{OH(max)}}}{I_{\text{IH(max)}}}\right| = \frac{400\ \mu\text{A}}{40\ \mu\text{A}} = \mathbf{10}$$

For the LOW output state, fan-out is calculated as follows: $I_{\text{OL(max)}}$ is the maximum current that the gate can sink. Each load input produces an $I_{\text{IL(max)}}$ of −1.6 mA. The LOW-state fan-out is

$$\left|\frac{I_{\text{OL(max)}}}{I_{\text{IL(max)}}}\right| = \frac{16\text{ mA}}{1.6\text{ mA}} = \mathbf{10}$$

In this case both the HIGH-state fan-out and the LOW-state fan-out are the same.

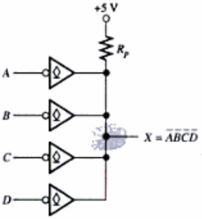
Supplementary Problem Determine the fan-out for a 74LS00 NAND gate.

Using Open-Collector Gates for Wired-AND Operation

The outputs of open-collector gates can be wired together to form what is called a *wired-AND* configuration. Figure 11–44 illustrates how four inverters are connected to produce a 4-input negative-AND gate. A single external pull-up resistor, R_p, is required in all wired-AND circuits.

▶ FIGURE 11–44

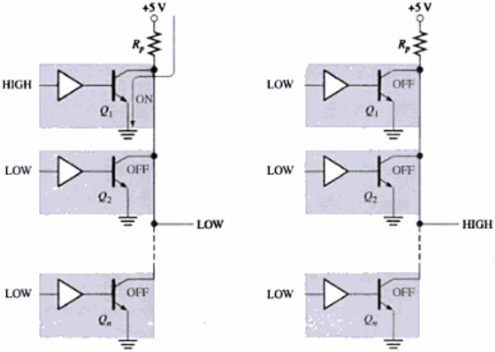
A wired-AND configuration of four inverters


+5 V
R_p
A
B
C
D
$X = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

When one (or more) of the inverter inputs is HIGH, the output X is pulled LOW because an output transistor is on and acts as a closed switch to ground, as illustrated in Figure 11–45(a). In this case only one inverter has a HIGH input, but this is sufficient to pull the output LOW through the saturated output transistor Q_1 as indicated.

▶ FIGURE 11–45

Open-collector wired negative-AND operation with inverters



(a) When one or more output transistors are on, the output is LOW.

(b) When all output transistors are off, the output is HIGH.

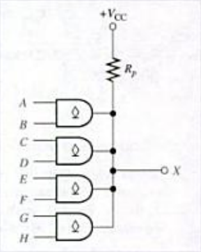
For the output X to be HIGH, *all* inverter inputs must be LOW so that all the open-collector output transistors are off, as indicated in Figure 11–45(b). When this condition exists, the output X is pulled HIGH through the pull-up resistor. Thus, the output X is HIGH only when *all* the inputs are LOW. Therefore, we have a negative-AND function, as expressed in the following equation:

$$X = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

EXAMPLE 11–7

Write the output expression for the wired-AND configuration of open-collector AND gates in Figure 11–46.

▶ FIGURE 11–46



Solution The output expression is

$$X = ABCDEFGH$$

The wired-AND connection of the four 2-input AND gates creates an 8-input AND gate.

Supplementary Problem Determine the output expression if NAND gates are used in Figure 11–46.

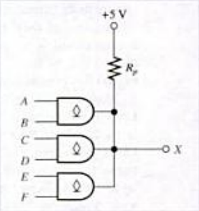
EXAMPLE 11–8

Three open-collector AND gates are connected in a wired-AND configuration as shown in Figure 11–47. Assume that the wired-AND circuit is driving four standard TTL inputs (-1.6 mA each).

(a) Write the logic expression for X.

(b) Determine the minimum value of R_p if $I_{OL(max)}$ for each gate is 30 mA and $V_{OL(max)}$ is 0.4 V.

▶ FIGURE 11–47



SECTION 11-6 REVIEW

1. What is a BiCMOS circuit?
2. In general, what is the main advantage of CMOS over bipolar (TTL)?

11-7 EMITTER-COUPLED LOGIC (ECL) CIRCUITS

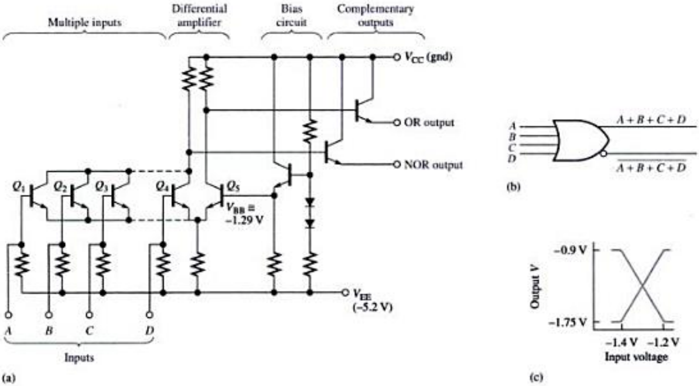
Emitter-coupled logic, like TTL, is a bipolar technology. The typical ECL circuit consists of a different amplifier input circuit, a bias circuit, and emitter-follower outputs. ECL is much faster than TTL because the transistors do not operate in saturation and is used in more specialized high-speed applications.

After completing this section, you should be able to

- Describe how ECL differs from TTL and CMOS
- Explain the advantages and disadvantages of ECL

An **ECL OR/NOR gate** is shown in Figure 11–53(a). The emitter-follower outputs provide the OR logic function and its NOR complement, as indicated by Figure 11–53(b).

Because of the low output impedance of the emitter-follower and the high input impedance of the differential amplifier input, high fan-out operation is possible. In this type of circuit, saturation is not possible. The lack of saturation results in higher power consumption and limited voltage swing (less than 1 V), but it permits high-frequency switching.


Multiple inputs
Differential amplifier
Bias circuit
Complementary outputs
V_{CC} (gnd)
OR output
NOR output
Q_1
Q_2
Q_3
Q_4
Q_5
V_{BB} = −1.29 V
V_{EE} (−5.2 V)
A
B
C
D
Inputs
(a)
A
B
C
D
A + B + C + D
$\overline{A+B+C+D}$
(b)
−0.9 V
−1.75 V
Output V
−1.4 V
−1.2 V
Input voltage
(c)

▲ **FIGURE 11–53**

An ECL OR/NOR gate circuit

The V_{CC} pin is normally connected to ground, and the V_{EE} pin is connected to −5.2 V from the power supply for best operation. Notice that in Figure 11–53(c) the output varies from a LOW level of −1.75 V to a HIGH level of −0.9 V with respect to ground. In positive logic a 1 is the HIGH level (less negative), and a 0 is the LOW level (more negative).

Noise Margin

As you have learned, the noise margin of a gate is the measure of its immunity to undesired voltage fluctuations (noise). Typical ECL circuits have noise margins from about 0.2 V to 0.25 V. These are less than for TTL and make ECL less suitable in high-noise environments.

Comparison of ECL with TTL and CMOS

Table 11-2 shows a comparison of key performance parameters for F and AHC with those for two ECL series, 10H and E-lite.

▶ TABLE 11-2

Comparison of two ECL series performance parameters

	BIPOLAR (TTL)	CMOS	BIPOLAR (ECL)	
	F	AHC	10H	E-LITE
Speed				
Gate propagation delay, t_p (ns)	3.3	3.7	1	0.22
FF maximum clock freq. (MHz)	145	170	330	2800
Power Dissipation Per Gate				
Bipolar: 50% dc	8.9 mW		25 mW	73 mW
CMOS: quiescent		2.5 μW		

SECTION 11-7 REVIEW

1. What is the primary advantage of ECL over TTL?
2. Name two disadvantages of ECL compared with TTL.

11-8 PMOS, NMOS, AND E^2CMOS

The PMOS and NMOS circuits are used largely in LSI functions, such as long shift registers, large memories, and microprocessor products. Such use is a result of the low power consumption and very small chip area required for MOS transistors. E^2CMOS is used in reprogrammable PLDs.

After completing this section, you should be able to

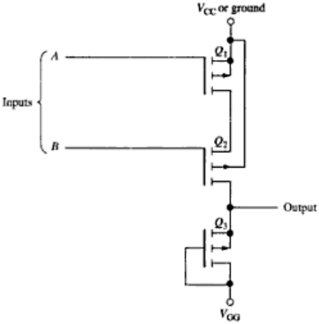
- Describe a basic PMOS gate
- Describe a basic NMOS gate
- Describe a basic E^2CMOS cell

PMOS

One of the first high-density **MOS** circuit technologies to be produced was **PMOS**. It utilizes enhancement-mode *p*-channel MOS transistors to form the basic gate building blocks. Figure 11–54 shows a basic PMOS gate that produces the NOR function in positive logic.

▶ **FIGURE 11–54**

Basic PMOS gate



The operation of the PMOS gate is as follows: The supply voltage V_{GG} is a negative voltage, and V_{CC} is a positive voltage or ground (0 V). Transistor Q_3 is permanently biased to create a constant drain-to-source resistance. Its sole purpose is to function as a current-limiting resistor. If a HIGH (V_{CC}) is applied to input A or B, then Q_1 or Q_2 is off, and the output is pulled down to a voltage near V_{GG}, which represents a LOW. When a LOW voltage (V_{GG}) is applied to both input A and input B, both Q_1 and Q_2 are turned on. This causes the output to go to a HIGH level (near V_{CC}). Since a LOW output occurs when either or both inputs are HIGH, and a HIGH output occurs only when all inputs are LOW, we have a NOR gate.

NMOS

The NMOS devices were developed as processing technology improved. The *n*-channel MOS transistor is used in NMOS circuits, as shown in Figure 11–55 for a NAND gate and a NOR gate.

In Figure 11–55(a), Q_3 acts as a resistor to limit current. When a LOW (V_{GG} or ground) is applied to one or both inputs, then at least one of the transistors (Q_1 or Q_2) is off, and the output is pulled up to a HIGH level near V_{CC}. When HIGHs (V_{CC}) are applied to both A and B, both Q_1 and Q_2 conduct, and the output is LOW. This action, of course, identifies this circuit as a NAND gate.

In Figure 11–55(b), Q_3 again acts as a resistor. A HIGH on either input turns Q_1 or Q_2 on, pulling the output LOW. When both inputs are LOW, both transistors are off, and the output is pulled up to a HIGH level.

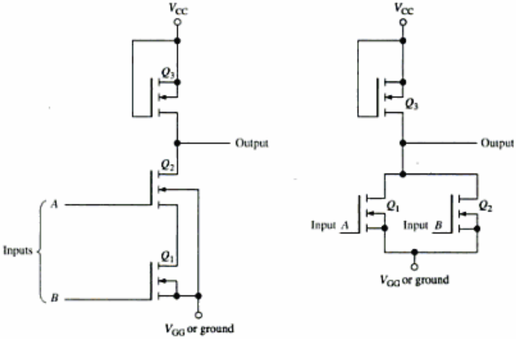
E^{2}CMOS

E^{2}CMOS technology is based on a combination of CMOS and NMOS technologies and is used in programmable logic devices such as GALs and CPLDs. An E^{2}CMOS cell is built around a MOS transistor with a floating gate that is externally charged or discharged by a small programming current. A schematic of this type of cell is shown in Figure 11–56.

When the floating gate is charged to a positive potential by removing electrons, the sense transistor is turned on, storing a binary zero. When the floating gate is charged to a negative potential by placing electrons on it, the sense transistor is turned off, storing a binary 1. The control gate controls the potential of the floating gate. The pass transistor isolates the sense transistor from the array during read and write operations that use the word and bit lines.

▶ FIGURE 11–55

Two NMOS gates

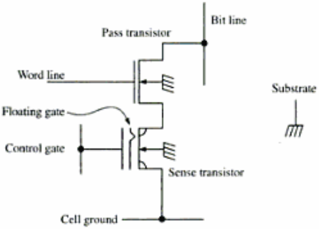

V_{CC}
Q_3
Output
Q_2
A
Inputs
Q_1
B
V_{GG} or ground
V_{CC}
Q_3
Output
Q_1
Q_2
Input A
Input B
V_{GG} or ground

(a) NAND

(b) NOR

▶ FIGURE 11–56

An E^{2}CMOS cell


Bit line
Pass transistor
Word line
Substrate
Floating gate
Control gate
Sense transistor
Cell ground

The cell is programmed by applying a programming pulse to either the control gate or the bit line of a cell that has been selected by a voltage on the word line. During the programming cycle, the cell is first erased by applying a voltage to the control gate to make the floating gate negative. This leaves the sense transistor in the *off* state (storing a 1). A write pulse is applied to the bit line of a cell in which a 0 is to be stored. This will charge the floating gate to a point where the sense transistor is on (storing a 0). The bit stored in the cell is read by sensing presence or absence of a small cell current in the bit line. When a 1 is stored, there is no cell current because the sense transistor is off. When a 0 is stored, there is a small cell current because the sense transistor is on. Once a bit is stored in a cell, it will remain indefinitely unless the cell is erased or a new bit is written into the cell.

SECTION 11–8 REVIEW

1. What is the main feature of NMOS and PMOS technology in integrated circuits?
2. What is the mechanism for charge storage in an E^{2}CMOS cell?

SUMMARY

- The categories of ICs in terms of circuit complexity are SSI (small-scale integration), MSI (medium-scale integration), LSI, VLSI, and ULSI (large-scale, very large-scale, and ultra large-scale integration).
- TTL is made with bipolar junction transistors.
- CMOS is made with MOS field-effect transistors.
- As a rule, CMOS has the lowest power consumption, TTL is next, and ECL has the highest power consumption.
- The average power dissipation of a logic gate is

$$P_D = V_{CC}\left(\frac{I_{CCH} + I_{CCL}}{2}\right)$$

- The speed-power product of a logic gate is

$$SPP = t_p P_D$$

- High-level noise margin of a gate is

$$V_{NH} = V_{OH(min)} - V_{IH(min)}$$

- Low-level noise margin of a gate is

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

- Average dc supply current for a gate is

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2}$$

- Power dissipation of a gate is

$$P_D = V_{CC}I_{CC}$$

- Totem-pole outputs of TTL cannot be connected together.
- Open-collector and open-drain outputs can be connected for wired-AND.
- A TTL device is not as vulnerable to electrostatic discharge (ESD) as is a CMOS device.
- Because of ESD, CMOS devices must be handled with great care.
- ECL is the fastest type of logic circuit.
- E^2CMOS is used in GALs and other PLDs.

SELF-TEST

Answers are at the end of the chapter.

1. A fixed-function IC package containing four AND gates in an example of

(a) MSI (b) SMT (c) SOIC (d) SSI

2. An LSI device has a circuit complexity of

(a) 12 to 99 equivalent gates (b) 100 to 9999 equivalent gates

(c) 2000 to 5000 equivalent gates (d) 10,000 to 99,999 equivalent gates

3. A positive-going pulse is applied to an inverter. The time interval from the leading edge of the input to the leading edge of the output is 7 ns. This parameter is

(a) speed-power product (b) propagation delay, t_{PLH}

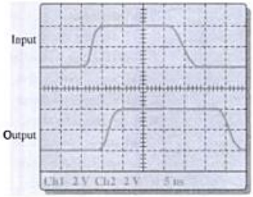
(c) propagation delay, t_{PLH} (d) pulse width

4. The CMOS family with the fastest switching speed is

(a) AC (b) HC (c) ACT (d) ALVC

5. Determine t_{PLH} and t_{PHL} from the oscilloscope display in Figure 11–57. The readings indicate V/div and sec/div for each channel.

FIGURE 11–57



6. Gate *A* has $t_{PLH} = t_{PHL} = 6$ ns. Gate *B* has $t_{PLH} = t_{PHL} = 10$ ns. Which gate can be operated at a higher frequency?
7. If a logic gate operates on a dc supply voltage of +5 V and draws an average current of 4 mA, what is its power dissipation?
8. The variable I_{CCH} represents the dc supply current from V_{CC} when all outputs of an IC are HIGH. The variable I_{CCL} represents the dc supply current when all outputs are LOW. For a 74LS00 IC, determine the typical power dissipation when all four gate outputs are HIGH. (See data sheet in Figure 11–6.)

SECTION 11–2 Basic Operational Characteristics and Parameters

9. A certain logic gate has a $V_{OH(min)} = 2.2$ V, and it is driving a gate with a $V_{IH(min)} = 2.5$ V. Are these gates compatible for HIGH-state operation? Why?
10. A certain logic gate has a $V_{OL(max)} = 0.45$ V, and it is driving a gate with a $V_{IL(max)} = 0.75$ V. Are these gates compatible for LOW-state operation? Why?
11. A TTL gate has the following actual voltage level values: $V_{IH(min)} = 2.25$ V, $V_{IL(max)} = 0.65$ V. Assuming it is being driven by a gate with $V_{OH(min)} = 2.4$ V and $V_{OL(max)} = 0.4$ V, what are the HIGH- and LOW-level noise margins?
12. What is the maximum amplitude of noise spikes that can be tolerated on the inputs in both the HIGH state and the LOW state for the gate in Problem 11?
13. Voltage specifications for three types of logic gates are given in Table 11–3. Select the gate that you would use in a high-noise industrial environment.

TABLE 11–3

	$V_{OH(MIN)}$	$V_{OL(MAX)}$	$V_{IH(MIN)}$	$V_{IL(MAX)}$
Gate *A*	2.4 V	0.4 V	2 V	0.8 V
Gate *B*	3.5 V	0.2 V	2.5 V	0.6 V
Gate *C*	4.2 V	0.2 V	3.2 V	0.8 V

14. A certain gate draws a dc supply current from a +5 V source of 2 mA in the LOW state and 3.5 mA in the HIGH state. What is the power dissipation in the LOW state? What is the power dissipation in the HIGH state? Assuming a 50% duty cycle, what is the average power dissipation?
15. Each gate in the circuit of Figure 11–58 has a t_{PLH} and a t_{PHL} of 4 ns. If a positive-going pulse is applied to the input as indicated, how long will it take the output pulse to appear?

SECTION 11–3 CMOS Circuits

21. Determine the state (on or off) of each MOSFET in Figure 11–61.
22. The CMOS gate network in Figure 11–62 is incomplete. Indicate the changes that should be made.
23. Devise a circuit, using appropriate CMOS logic gates and/or inverters, with which signals from four different sources can be connected to a common line at different times without interfering with each other.

FIGURE 11–61

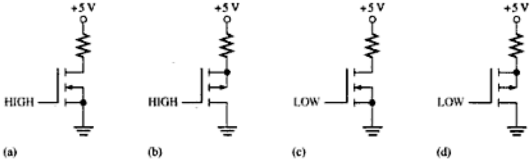
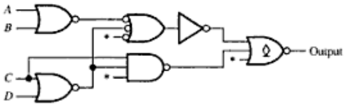

+5 V
+5 V
+5 V
+5 V
HIGH
HIGH
LOW
LOW
(a)
(b)
(c)
(d)

FIGURE 11–62

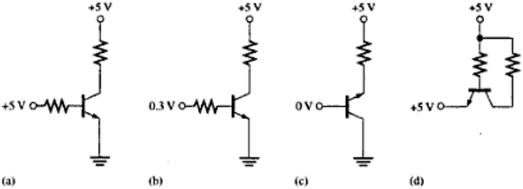

A
B
C
D
*
*
*
Output

* unused inputs

SECTION 11–4 TTL Circuits

24. Determine which BJTs in Figure 11–63 are off and which are on.

FIGURE 11–63


+5 V
+5 V
+5 V
+5 V
+5 V
0.3 V
0 V
+5 V
(a)
(b)
(c)
(d)

25. Determine the output state of each TTL gate in Figure 11–64.
26. The TTL gate network in Figure 11–65 is incomplete. Indicate the changes that should be made.

FIGURE 11–64

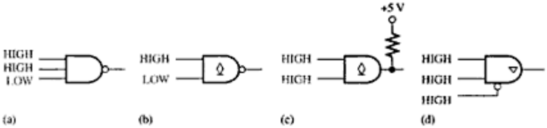
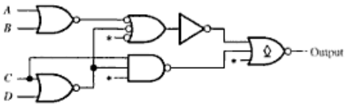

HIGH
HIGH
LOW
HIGH
LOW
+5 V
HIGH
HIGH
HIGH
HIGH
HIGH
(a)
(b)
(c)
(d)

FIGURE 11–65


A
B
C
D
Output

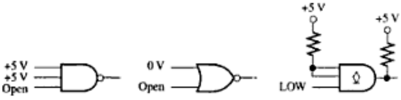
* unused inputs

SECTION 11–5 Practical Considerations in the Use of TTL

27. Determine the output level of each TTL gate in Figure 11–66.

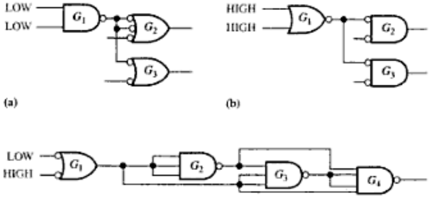
28. For each part of Figure 11–67, tell whether each driving gate is sourcing or sinking current. Specify the maximum current out of or into the output of the driving gate or gates in each case. All gates are standard TTL.

FIGURE 11–66


+5 V
+5 V
Open
0 V
Open
+5 V
+5 V
LOW

(a) (b) (c)

FIGURE 11–67


LOW
LOW
G_1
G_2
G_3
HIGH
HIGH
G_1
G_2
G_3
LOW
HIGH
G_1
G_2
G_3
G_4

(a) (b)

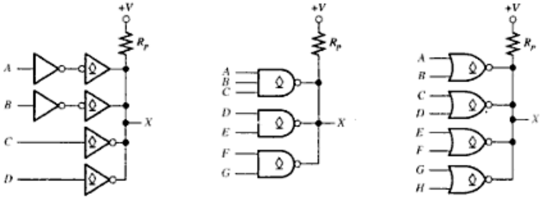
(c)

29. Use open-collector inverters to implement the following logic expressions:

(a) $X = \overline{A}\,\overline{B}\,\overline{C}$ (b) $X = A\overline{B}C\overline{D}$ (c) $X = ABC\overline{D}\,\overline{E}\,\overline{F}$

30. Write the logic expression for each of the circuits in Figure 11–68.

FIGURE 11–68


+V
R_p
A
B
C
D
X
+V
R_p
A
B
C
D
E
F
G
X
+V
R_p
A
B
C
D
E
F
G
H
X

(a) (b) (c)

31. Determine the minimum value for the pull-up resistor in each circuit in Figure 11–68 if $I_{OL(max)} = 40$ mA and $V_{OL(max)} = 0.25$ V for each gate. Assume that 10 standard TTL unit loads are being driven from output X and the supply voltage is 5 V.

32. A certain relay requires 60 mA. Devise a way to use open-collector NAND gates with $I_{OL(max)} = 40$ mA to drive the relay.

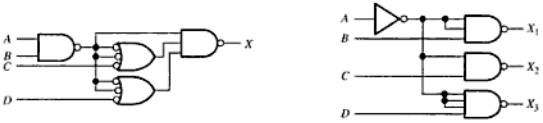
SECTION 11–6 Comparison of CMOS and TTL Performance

33. Select the IC family with the best speed-power product in Table 11–1.

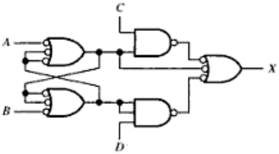
34. Determine from Table 11–1 the logic family that is most appropriate for each of the following requirements:
    - (a) shortest propagation delay time
    - (b) fastest flip-flop toggle rate
    - (c) lowest power dissipation
    - (d) best compromise between speed and power for a logic gate

35. Determine the total propagation delay from each input to each output for each circuit in Figure 11–69.

FIGURE 11–69



(a) 74FXX gates (b) 74HCXX gates



(c) 74AHCXX gates

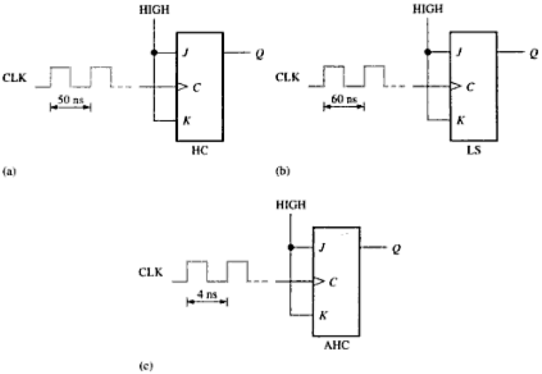
36. One of the flip-flops in Figure 11–70 may have an erratic output. Which one is it if any and why?

SECTION 11–7 Emitter-Coupled Logic (ECL) Circuits

37. What is the basic difference between ECL circuitry and TTL circuitry?

38. Select ECL, HCMOS, or the appropriate TTL series for each of the following requirements:
    - (a) highest speed
    - (b) lowest power
    - (c) best compromise between high speed and low power (speed-power product)

FIGURE 11-70



ANSWERS

SECTION REVIEWS

SECTION 11–1 Basics of Digital Integrated Circuits

1. SSI—small-scale integration; MSI—medium-scale integration; LSI—large-scale integration; VLSI—very large-scale integration; ULSI—ultra large-scale integration
2. **(a)** MSI **(b)** LSI **(c)** SSI **(d)** VLSI **(e)** ULSI
3. TTL, CMOS, and ECL; CMOS and TTL
4. **(a)** LS—Low-power Schottky **(b)** ALS—Advanced LS
   (c) F—fast TTL **(d)** HC—High-speed CMOS
   (e) AC—Advanced CMOS **(f)** HCT—HC CMOS TTL compatible
   (g) LV—Low-voltage CMOS
5. **(a)** 74LS04—Hex inverter **(b)** 74HC00—Quad 2-input NAND
   (c) 74LV08—Quad 2-input AND **(d)** 74ALS10—Triple 3-input NAND
   (e) 7432—Quad 2-input OR **(f)** 74ACT11—Triple 3-input AND
   (g) 74AHC02—Quad 2-input NOR
6. Lowest power—CMOS
7. Six inverters in a package; four 2-input NAND gates in a package
8. $t_{PLH} = 10$ ns; $t_{PHL} = 8$ ns
9. 18 pJ
10. I_{CCL}—dc supply current for LOW output state; I_{CCH}—dc supply current for HIGH output state
11. V_{IL}—LOW input voltage; V_{IH}—HIGH input voltage
12. V_{OL}—LOW input voltage; V_{OH}—HIGH output voltage

SECTION 11–2 Basic Operational Characteristics and Parameters

1. V_{IH}: HIGH level input voltage: V_{IL}: LOW level input voltage; V_{OH}: HIGH level output voltage; V_{OL}: LOW level output voltage
2. A higher value of noise margin is better.
3. Gate *B* can operate at a higher frequency.
4. Excessive loading reduces the noise margin of a gate.

SECTION 11–3 CMOS Circuits

1. MOSFETs are used in CMOS logic.
2. A complementary output circuit consists of an *n*-channel and a *p*-channel MOSFET.
3. Because electrostatic discharge can damage CMOS devices.

SECTION 11–4 TTL Circuits

1. False, the *npn* BJT is off.
2. The *on* state of a BJT is a closed switch; the *off* state is an open switch.
3. Totem-pole and open-collector are types of TTL outputs.
4. Tristate logic provides a high-impedance state, in which the output is disconnected from the rest of the circuit.

SECTION 11–5 Practical Considerations in the Use of TTL

1. Sink current occurs in a LOW output state.
2. Source current is less than sink current because a TTL load looks like a reverse-biased diode in the HIGH state.
3. The totem-pole transistors cannot handle the current when one output tries to go HIGH and the other is LOW.
4. Wired-AND must use open-collector.
5. Lamp driver must be open-collector.
6. False, an unconnected TTL input generally acts as a HIGH.

SECTION 11–6 Comparison of CMOS and TTL Performance

1. BiCMOS uses bipolar transistors for input and output circuitry and CMOS in between.
2. CMOS has lower power dissipation than bipolar.

SECTION 11–7 Emitter-coupled Logic (ECL) Circuits

1. ECL is faster than TTL.
2. ECL has more power and less noise margin than TTL.

SECTION 11–8 PMOS, NMOS, and E^2CMOS

1. NMOS and PMOS are high density.
2. The floating gate is the mechanism for storing charge in an E^2CMOS cell.

SUPPLEMENTARY PROBLEMS FOR EXAMPLES

11–1 The gate with 4 ns t_{PLH} and t_{PHL} can operate at the highest frequency.

11–2 10 mW

11–3 CMOS

11–4 10.75 μW

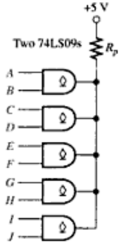
11–5 $I_{T(source)} = 5(20\ \mu A) = 100\ \mu A$
$I_{T(sink)} = 5(-0.4\ mA) = -2.0\ mA$

11–6 Fan-out = 20

11–7 $X = (\overline{AB})(\overline{CD})(\overline{EF})(\overline{GH}) = (\overline{A} + \overline{B})(\overline{C} + \overline{D})(\overline{E} + \overline{F})(\overline{G} + \overline{H})$

11–8 See Figure 11–71.

▶ **FIGURE 11–71**



11–9 $R_L = 97\ \Omega$

SELF-TEST

1. (d) **2.** (d) **3.** (b) **4.** (d) **5.** (d) **6.** (b) **7.** (c) **8.** (c)
9. (c) **10.** (e) **11.** (a) **12.** (c) **13.** (d) **14.** (a) **15.** (c)

12

PROGRAMMABLE LOGIC DEVICES (PLDs)

CHAPTER OBJECTIVES

- Describe PLD, discuss the various types, and state how PLDs are programmed
- Describe the basic concepts of programmable logic
- Describe how a PAL works and explain how the part number defines its configuration
- Describe a GAL and how it differs from a PAL
- Describe basically how SPLDs are programmed
- Describe basic architecture of GAL22V10 and GAL16V8
- Describe a basic CPLD
- Describe FPGA and explain how it differs from a CPLD

INTRODUCTION

So far, you have learnt fixed-function ICs. These ICs cover a wide range of gates, combinational circuits and sequential circuits. The fixed-function ICs perform the designated function, which cannot be altered or programmed for performing any other logic function. Another category of ICs are programmable, which can be programmed by a user for performing variety of functions. Simple programmable logic devices (SPLDs) such as Programmable Array Logic (PAL) and Generic Array Logic (GAL) have been discussed in detail. The Complex programmable logic devices (CPLDs) and the Field-programmable Gate Arrays (FPGAs) have also been introduced.

12–1 INTRODUCTION TO PROGRAMMABLE LOGIC DEVICES (PLDs)

You have learned the various categories (SSI, MSI, LSI, VLSI, and ULSI) of certain fixed-function logic circuits that are available and you saw some of the packaging configurations. In fixed-function devices, a specific logic function is contained in the IC package when it is purchased and it can never be changed.

Another category of logic device is one in which the logic function is programmed by the user and, in some cases, can be reprogrammed many times. These devices are called *programmable logic devices* or PLDs. This section is a brief introduction to PLDs and how they compare to fixed-function logic devices. In the following chapters, PLDs are covered in detail.

After competing this section, you should be able to

■ State the major types of PLDs ■ Explain the difference between PLDs and fixed-function devices (SSI and MSI) ■ State the advantages of PLDs over SSI and MSI logic devices (fixed-function) ■ Describe two ways in which a logic function can be programmed into a PLD

In many applications the **PLD** has replaced the hard-wired fixed-function logic device. You can expect to see a continued growth in PLDs. However, fixed-function logic is still important and will be around for a long time but in more limited applications. One area in which fixed-function logic is very effective is in the laboratory for teaching basic concepts.

One advantage of PLDs over fixed-function logic devices is that many more logic circuits can be "stuffed" into a much smaller area with PLDs. A second advantage is that, with certain PLDs, logic designs can be readily changed without rewiring or replacing components. A third advantage is that, generally, a PLD design can be implemented faster than one using fixed-function ICs once the required programming language is mastered.

Types of PLDs

The three major types of programmable logic are SPLD, CPLD, and FPGA. Each major type generally has several manufacturer-specific subcategories.

SPLDs (simple programmable logic devices) are the least complex form of PLDs. An SPLD can typically replace several fixed-function SSI or MSI devices and their interconnections. The SPLD was the first type of programmable logic available. A few categories of SPLD are listed below, some of which are unique to a specific manufacturer. A typical package has 24 to 28 pins, and one is shown in Figure 12–1.

- PAL (programmable array logic)
- GAL (generic array logic)
- PLA (programmable logic array)
- PROM (programmable read-only memory)

FIGURE 12–1

Typical SPLD package

CPLDs (complex programmable logic devices) have a much higher capacity than SPLDs, permitting more complex logic circuits to be programmed into them. A typical CPLD is the equivalent of from two to sixty-four SPLDs. The development of these devices followed the SPLD as advances in technology permitted higher-density chips to be implemented. There are several forms of CPLD, which vary in complexity and programming capability. CPLDs typically come in 44-pin to 160-pin packages depending on the complexity. Examples of CPLD packages are shown in Figure 12–2.

FIGURE 12–2

Typical CPLD packages

(a) 84-pin PLCC package (b) 128-pin PQFP package

FPGAs (field-programmable gate arrays) are different from SPLDs and CPLDs in their internal organization and have the greatest logic capacity. FPGAs consist of an array of anywhere from sixty-four to thousands of logic-gate groups that are sometimes called *logic blocks*. Two basic classes of FPGA are *course-grained* and *fine-grained*. The course-grained FPGA has large logic blocks, and the fine-grained FPGA has much smaller logic blocks. FPGAs come in packages ranging up to 1000 pins or more.

PLD Programming

A logic circuit design for a PLD is entered using one of two basic methods: schematic entry or text-based entry. Sometimes, a combination of both methods is used.

In the schematic entry method, the software allows the user to enter a logic design using logic components (e.g. logic gates, flip-flops) and to interconnect them on the computer screen to form a schematic diagram.

In the text-based entry method, also known as *language-based* entry, the software allows the user to enter a logic design in the form of text using a **hardware description language (HDL)**. Several HDLs are available, such as VHDL and Verilog HDL developed for programming PLDs and are widely used.

An HDL that is becoming widely used, especially for programming CPLDs and FPGAs, is VHDL, a standard developed by the Department of Defense and adopted by the IEEE (Institute of Electrical and Electronics Engineers). The latest version of VHDL is IEEE std. 1076–1993. Verilog is another popular HDL for programming CPLDs and FPGAs. In addition, there are several proprietary HDLs provided by various manufacturers for their products.

SECTION 12–1 REVIEW

1. What does PLD stand for?
2. What does SPLD stand for?
3. What does CPLD stand for?
4. What does FPGA stand for?
5. Basically, how does a CPLD differ from an SPLD?
6. List two ways in which a logic design can be entered for PLD programming.
7. What does HDL mean?
8. Name two HDLs that were developed for PLDs.
9. Name an important IEEE standard HDL.

12–2 SIMPLE PROGRAMMABLE LOGIC DEVICES (SPLDs)

Programmable logic devices (PLDs) are used in many applications to replace fixed-function circuits; they save space and reduce the actual number and cost of devices in a given design. An **SPLD** (simple programmable logic device) consists of an array of AND gates and OR gates that can be programmed to achieve specified logic functions. Four types of devices that are classified as SPLDs are the programmable read-only memory (PROM), the programmable logic array (PLA), the programmable array logic (PAL), and the generic array logic (GAL).

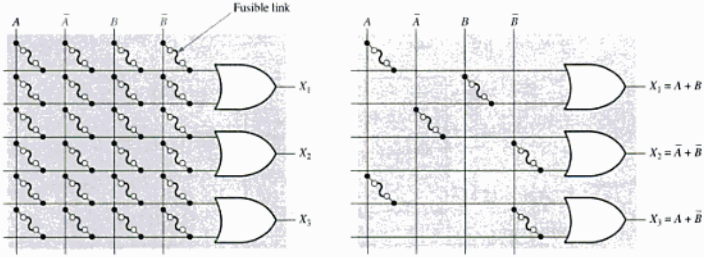
After completing this section, you should be able to

- Explain a basic OR array
- Explain a basic AND array
- Describe the structure of a PROM
- Describe the structure of a PLA
- Describe the basic PAL
- Describe the basic GAL
- Discuss the differences between a PAL and a GAL

Programmable Arrays

All PLDs consist of programmable arrays. A programmable **array** is essentially a grid of conductors that form rows and columns with a fusible link at each cross point. Arrays can be either fixed or programmable. The earliest type of programmable array, dating back to the 1960s, was a matrix with a diode at each cross point of the matrix.

The OR Array The original diode array evolved into the integrated OR array, which consists of an array of OR gates connected to a programmable matrix with fusible links at each cross point of a row and column, as shown in Figure 12–3(a). The array can be programmed by blowing **fuses** to eliminate selected variables from the output functions, as illustrated in part (b) for a specific case. For each input to an OR gate, only one fuse is left intact in order to connect the desired variable to the gate input. Once a fuse is blown, it cannot be reconnected.



(a) Unprogrammed

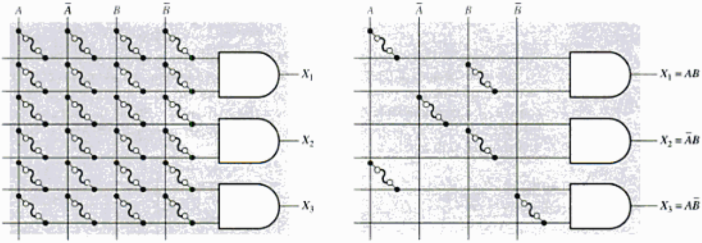
(b) Programmed

▲ **FIGURE 12–3**

An example of a basic programmable OR array

Another method of programming a PLD is the antifuse, which is the opposite of the fuse. Instead of a fusible link being broken or opened to program a variable, a normally open contact is shorted by "melting" the antifuse material to form a connection.

The AND Array This type of array consists of AND gates connected to a programmable matrix with fusible links at each cross point, as shown in Figure 12–4(a). Like the OR array, the AND array can be programmed by blowing fuses to eliminate variables from the output function, as illustrated in part (b). For each input to an AND gate, only one fuse is left intact in order to connect the desired variable to the gate input. Also, like the OR array, the AND array with fusible links or with antifuses is one-time programmable.



(a) Unprogrammed

(b) Programmed

▲ **FIGURE 12–4**

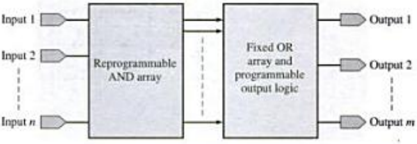
An example of a basic programmable AND array

***Generic Array Logic** (GAL)* The GAL has a reprogrammable AND array and a fixed OR array with programmable output logic. The two main differences between GAL and PAL devices are (a) the GAL is *reprogrammable* and (b) the GAL has programmable output configurations.

The GAL can be reprogrammed again and again because it uses **E^{2}CMOS** (electrically erasable CMOS) technology instead of bipolar technology and fusible links. The block diagram of a GAL is shown in Figure 12–8.

▶ **FIGURE 12–8**

Block diagram of a GAL



SECTION 12–2 REVIEW

1. List four SPLDs.
2. What is the difference between a PLA and a PAL?
3. Describe the differences between a PAL and a GAL?

12–3 PROGRAMMABLE ARRAY LOGIC (PAL)

The PAL is a type of SPLD that is one-time programmable (OTP). As you learned in the last chapter, the PAL in its basic form is an SPLD with an AND array and a fixed OR array. In this section, you will learn how PALs are used to produce specified combinational logic functions and examine a specific PAL.

After completing this section, you should be able to

- Describe basic PAL operation
- Show how a sum-of-products expression is implemented in a PAL
- Discuss simplified PAL logic diagrams
- Explain the three basic types of PAL output combinational logic
- Interpret standard PAL numbers
- Discuss the PAL16L8

PAL Operation

The **PAL** consists of a programmable array of AND gates that connects to a fixed array of OR gates. This structure allows any sum-of-products (SOP) logic expression with a defined number of variables to be implemented. As you have learned, any logic function can be expressed in SOP form.

The basic structure of a PAL is illustrated in Figure 12–9 for two input variables and one output although most PALs have many inputs and many outputs. As you know, a programmable array is essentially a grid of conductors forming rows and columns with a fusible link at each cross point. Each fused cross point of a row and column is called a **cell** and is the programmable element of a PAL. Each row is connected to the input of an AND gate and each column is connected to an input variable or its complement. By using the presence or absence of fused connections created by programming, any combination of input variables or complements can be applied to an AND gate to form any desired product term.

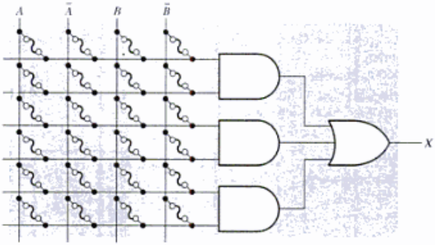


FIGURE 12–9

Basic structure of a PAL

Implementing a Sum-of-Products Expression In its simplest form, each cell in a basic AND array consists of a fusible link connecting a row and a column as represented in Figure 12–9. When the connection between a row and column is required, the fuse is left intact. When no connection between a row and column is required, the fuse is blown open during the programming process.

As an example, a simple array is programmed as shown in Figure 12–10 so that the product term AB is produced by the top AND gate, $A\overline{B}$ by the middle AND gate, and $\overline{A}\,\overline{B}$ by the bottom AND gate. As you can see, the fusible links are left intact to connect the desired variables or their complements to the appropriate AND gate inputs. The fusible links are opened where a variable or its complement is not used in a given product term. The final output from the OR gate is the SOP expression,

$$X = AB + A\overline{B} + \overline{A}\,\overline{B}$$

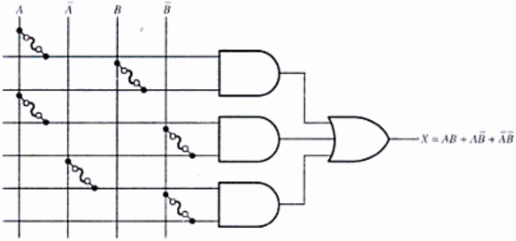


FIGURE 12–10

PAL implementation of a SOP expression

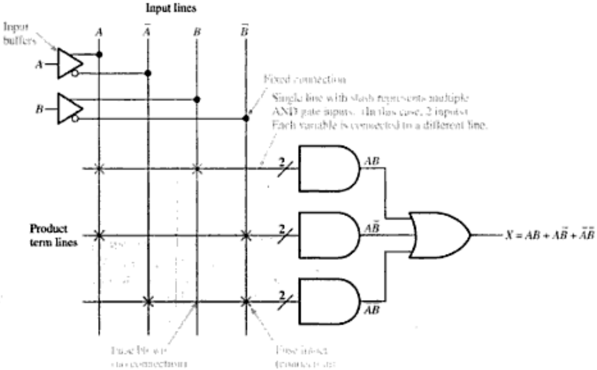
Simplified Symbols

What you have seen so far represents a small segment of a typical PAL. Actual PALs have many AND gates and many OR gates in addition to other circuitry and are capable of handling many input variables and their complements. Since PALs are very complex integrated circuit devices, manufacturers have adopted a simplified notation for the logic diagrams to keep them from being overwhelmingly complicated.

Input Buffers The input variables to a PAL are buffered to prevent loading by the large number of AND gate inputs to which a variable or its complement may be connected. An inverting buffer produces the complement of an input variable. The symbol representing the **buffer** circuit that produces both the variable and its complement on its outputs is shown in Figure 12–11 where the bubble output is the complement.

AND Gates A typical PAL AND array has an extremely large number of interconnecting lines, and each AND gate has multiple inputs. PAL logic diagrams show an AND gate that actually has several inputs by using an AND gate symbol with a single input line representing all of its input lines, as indicated in Figure 12–11. Multiple input lines are sometimes indicated by a slash and the number of lines as shown for the case of two lines.

PAL Connections To keep a logic diagram as simple as possible, the fusible links in a programmed AND array are indicated by an X at the cross point if the fuse is left intact and by the absence of an X if the fuse is blown, as indicated in Figure 12–11. Fixed connections use the standard dot notation, as also indicated.


Input lines
Input buffers
A
$\bar{A}$
B
$\bar{B}$
A
B
Fixed connection
Single line with slash represents multiple AND gate inputs. (In this case, 2 inputs.) Each variable is connected to a different line.
2
AB
$A\bar{B}$
$\bar{A}\bar{B}$
$X = AB + A\bar{B} + \bar{A}\bar{B}$
Product term lines
Fuse blown (no connection)
Fuse intact (connected)

▲ **FIGURE 12–11**

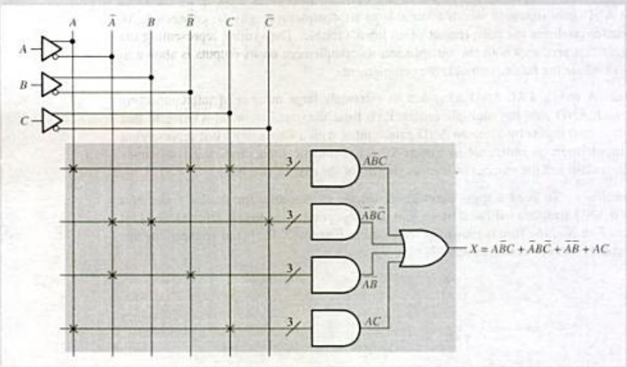
Simplified diagram of a programmed PAL

EXAMPLE 12–1

Show how a PAL is programmed for the following 3-variable logic function:

$$X = A\bar{B}C + \bar{A}B\bar{C} + \bar{A}\bar{B} + AC$$

Solution The programmed array is shown in Figure 12–12. The intact fusible links are indicated by small red Xs. The absence of an X means that the fuse has been blown.

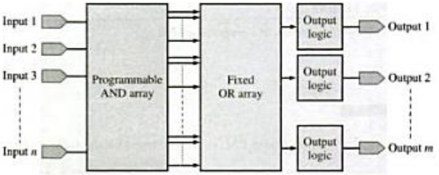

A
$\bar{A}$
B
$\bar{B}$
C
$\bar{C}$
A
B
C
3
3
3
3
$A\bar{B}C$
$\bar{A}B\bar{C}$
$\bar{A}\bar{B}$
AC
$X = A\bar{B}C + \bar{A}B\bar{C} + \bar{A}\bar{B} + AC$

▲ **FIGURE 12–12**

Supplementary Problem Write the expression for the output if the fusible links connecting input A to the top row and to the bottom row in Figure 12–12 are also blown.

The PAL Block Diagram

A block diagram of a PAL is shown in Figure 12–13. The AND array outputs go to the OR array, and the output of each OR gate goes to its associated output logic. A typical PAL has eight or more inputs to its AND array and up to eight outputs from its output logic as indicated, where $n \geq 8$ and $m \leq 8$. Some PALs provide a combined input and output (I/O) pin that can be programmed as either an output or an input. The symbol ⬢ means that a pin can be either an input or an output.


Input 1
Input 2
Input 3
Input n
Programmable AND array
Fixed OR array
Output logic
Output logic
Output logic
Output 1
Output 2
Output m

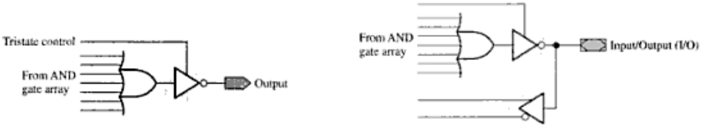
▲ **FIGURE 12–13**

Block diagram of a PAL

PAL Output Combinational Logic

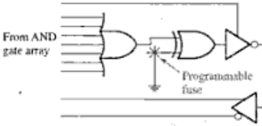
There are several basic types of PAL output logic that allows you to configure the device for a specific application. In this chapter, only the aspects of the output logic related to combinational logic functions are discussed. Additional features are covered later after you have studied flip-flops, counters, and registers. Figure 12–14 shows three basic types of combinational output logic with tristate outputs and the associated OR gate. The following are types of PAL output logic:

- *Combinational output* This output is used for an SOP function and is usually available as either an active-LOW or an active-HIGH output.
- *Combinational input/output (I/O)* This output is used when the output function must feed back to be an input to the array or be used to make the I/O pin an input only.
- *Programmable polarity output* This output is used for selecting either the output function or its complement by programming the exclusive-OR gate for inversion or noninversion. The fusible link on the exclusive-OR input is blown open for inversion and left intact for noninversion.



(a) Combinational output (active-LOW). An active-HIGH output would be shown without the bubble on the tristate gate symbol.

(b) Combinational input/output (active-LOW)



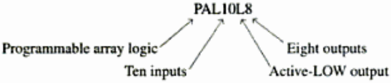
(c) Programmable polarity output

▲ **FIGURE 12–14**

Basic types of PAL combinational output logic

Standard PAL Numbering

Standard PALs come in a variety of configurations, each of which is identified by a unique part number. This part number always begins with the prefix PAL. The first two digits following PAL indicate the number of inputs, which includes outputs that can be configured as inputs. The letter following the number of inputs designates the type of output: L—active-LOW, H—active-HIGH, or P—programmable polarity. The one or two digits that follow the output type is the number of outputs. The following number is an example.


PAL10L8
Programmable array logic
Ten inputs
Active-LOW output
Eight outputs

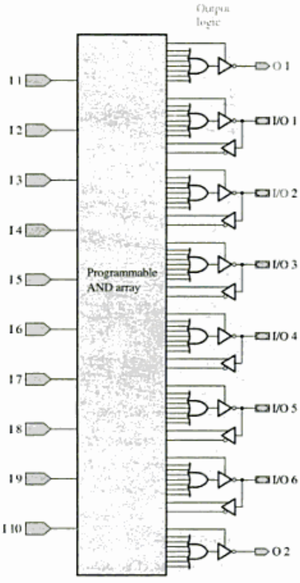
In addition, a PAL part number may carry suffixes that specify speed, package type, and temperature range.

A PAL

The PAL16L8 is an example of a PAL configuration, and its block diagram is shown in Figure 12–15. This particular device has 10 dedicated inputs (I), 2 dedicated outputs (O), and 6 pins that can be used either as inputs or as outputs. Each of the outputs is active-LOW.

▶ **FIGURE 12–15**

Block diagram of the PAL16L8


Output logic
I 1
I 2
I 3
I 4
I 5
I 6
I 7
I 8
I 9
I 10
Programmable AND array
O 1
I/O 1
I/O 2
I/O 3
I/O 4
I/O 5
I/O 6
O 2

I = input, O = output, I/O = input/output

EXAMPLE 12–2

Determine the number of inputs, the number of outputs, and the type of output for each of the following PAL part numbers:

(a) PAL12H6 **(b)** PAL16L2 **(c)** PAL20P8

Solution **(a)** 12 inputs, 6 outputs, active-HIGH outputs

(b) 16 inputs, 2 outputs, active-LOW outputs

(c) 20 inputs, 8 outputs, programmable polarity outputs

Supplementary Problem Describe a PAL with the part number PAL14H4.

SECTION 12–3 REVIEW

1. What is a PAL?
2. Explain the purpose of a fusible link in a PAL.
3. Can any combinational logic function be implemented with a PAL?
4. List three types of PAL output logic.
5. What does the designation PAL12H6 mean?

12–4 BASIC CONCEPTS OF GAL

GAL is a designation originally used by Lattice Semiconductor and later licensed to other manufacturers. The GAL in its basic form is a PAL with a reprogrammable AND array, a fixed OR array, and programmable output logic. In this section, basic GAL concepts are introduced. The details of GAL 22V10 and 16V8 devices will be discussed in Section 12–6 and 12–7, respectively.

After completing this section, you should be able to

▪ Describe basic GAL operation ▪ Show how a sum-of-products expression is implemented in a GAL ▪ Compare the E^2CMOS cell in a GAL with the fusible link cell in a PAL ▪ Define OLMC and explain its purpose ▪ Interpret standard GAL numbers

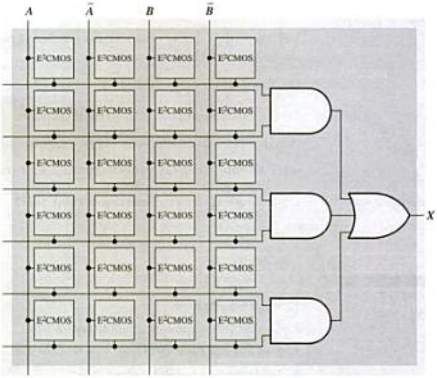
GAL Operation

The GAL basically consists of a reprogrammable array of AND gates that connects to a fixed array of OR gates. Just as in a PAL, this structure allows any sum-of-products (SOP) logic expression with a defined number of variables to be implemented.

The basic structure of a GAL is illustrated in Figure 12–16 for two input variables and one output although most GALs have many inputs and many outputs. The reprogrammable array is essentially a grid of conductors forming rows and columns with an electrically erasable CMOS (E^2CMOS) cell at each cross point, rather than a fuse as in a PAL. These cells are shown as blocks in the figure.

▲ **FIGURE 12–16**

Basic E^2CMOS structure of a GAL

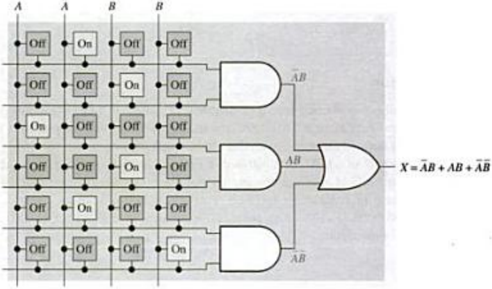

A
$\overline{A}$
B
$\overline{B}$
E²CMOS E²CMOS E²CMOS E²CMOS
E²CMOS E²CMOS E²CMOS E²CMOS
E²CMOS E²CMOS E²CMOS E²CMOS
E²CMOS E²CMOS E²CMOS E²CMOS
E²CMOS E²CMOS E²CMOS E²CMOS
E²CMOS E²CMOS E²CMOS E²CMOS
X

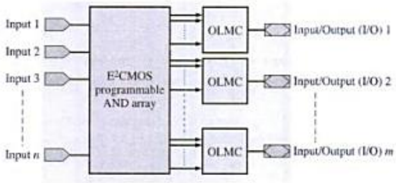
Each row is connected to the input of an AND gate, and each column is connected to an input variable or its complement. By programming each E^2CMOS cell to be either *on* or *off*, any combination of input variables or complements can be applied to an AND gate to form any desired product term. A cell that is *on* effectively connects its corresponding row and column, and a cell that is *off* disconnects the row and column. The cells can be electrically erased and reprogrammed. A typical E^2CMOS cell can retain its programmed state for 20 years or more. See Section 11.8 for a discussion of how an E^2CMOS cell works.

Implementing a Sum-of-Products Expression As an example, a simple GAL array is programmed as shown in Figure 12–17 so that the product term $\overline{A}B$ is produced by the top AND gate, AB by the middle AND gate, and $\overline{A}\,\overline{B}$ by the bottom AND gate. As shown, the E^2CMOS cells are *on* to connect the desired variables or their complements to the appropriate AND gate inputs. The E^2CMOS cells are *off* where a variable or its complement is not used in a given product term. The final output from the OR gate is an SOP expression.

► **FIGURE 12–17**

GAL implementation of a SOP


A
$\overline{A}$
B
$\overline{B}$
Off On Off Off
Off Off On Off
On Off Off Off
Off Off On Off
Off On Off Off
Off Off Off On
$\overline{A}B$
AB
$\overline{A}\,\overline{B}$
$X = \overline{A}B + AB + \overline{A}\,\overline{B}$

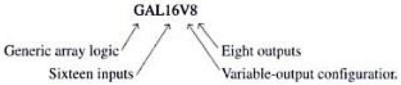

Input 1
Input 2
Input 3
Input n
E^2CMOS programmable AND array
OLMC
OLMC
OLMC
Input/Output (I/O) 1
Input/Output (I/O) 2
Input/Output (I/O) m

▲ **FIGURE 12–19**

Block diagram of a GAL

Standard GAL Numbering

GALs come in a variety of configurations, each of which is identified by a unique part number. This part number always begins with the prefix GAL. The first two digits following the prefix indicate the number of inputs, which includes outputs that can be configured as inputs. The letter V following the number of inputs designates a variable-output configuration. The one or two digits that follow the output type is the number of outputs. The following number is an example.


GAL16V8
Generic array logic
Sixteen inputs
Eight outputs
Variable-output configuration

EXAMPLE 12–4

Determine the number of inputs and the number of outputs for each of the following GAL part numbers:

(a) GAL20V8 **(b)** GAL22V10

Solution **(a)** 20 inputs, 8 outputs

(b) 22 inputs, 10 outputs

Supplementary Problem Describe a GAL with the part number GAL18V10.

SECTION 12–4 REVIEW

1. What is a GAL?
2. Explain the purpose of the E^2CMOS cell in a GAL.
3. Can any combinational logic function be implemented with a GAL?
4. What is the OLMC?

12–5 PROGRAMMING OF SPLDs

So far, you have learned about the basic internal organization (architecture) of some types of SPLDs, including the PAL and the GAL. In this section, the general concept of SPLD programming is discussed. From a hardware standpoint, programming generally falls into two major categories: the conventional method and the in-system programming (ISP) method. From a software standpoint, a logic design is programmed into a PLD using either schematic entry software, hardware description language (HDL) software, or a combination of both.

After completing this section, you should be able to

- Discuss the requirements for conventionally programming an SPLD
- Describe a JEDEC file
- Discuss the requirements for in-system programming (ISP) of an SPLD
- Discuss the JTAG standard

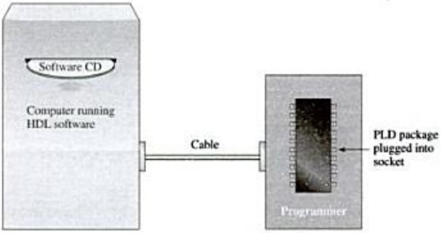
Conventional Programming of an SPLD

SPLDs are conventionally programmed using programming software, a computer, and a device programming fixture (programmer) connected to the computer. The programmer has a socket that accepts the SPLD package. With conventional programming, an SPLD is programmed while plugged into the socket of the programmer, as illustrated in Figure 12–20. This type of programming is for SPLDs that have not yet been installed on a printed circuit board.

The Computer Any PC that meets the software and programmer specifications can be used. These specifications normally include the type of microprocessor on which the computer is based, the amount of memory, and the operating system.

FIGURE 12–20

Typical configuration for conventional PLD programming


Software CD
Computer running HDL software
Cable
PLD package plugged into socket
Programmer

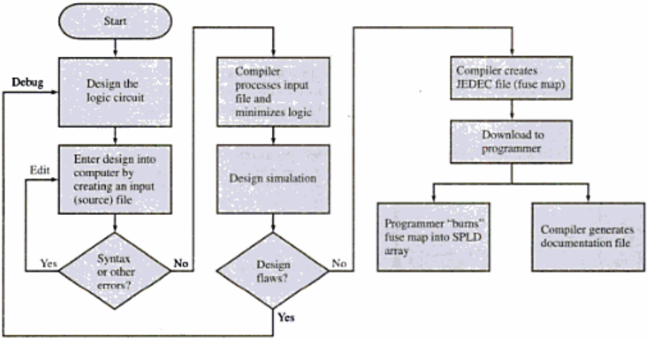
The Software The software packages for SPLD programming are called *logic compilers*. Several software packages are available for programming SPLDs. All of these software packages process and synthesize the logic design entered by a specified method, convert the entered data into an intermediate file, and then generate a final output file called a **JEDEC file** (also known as a *cell map* or *fuse map*). Also, the software provides for simulating and debugging a logic design before the design is finalized.

A key feature of the software package is the method(s) for entering a logic design. Most software packages provide several entry options for implementation of a logic design in an SPLD. Three text-based design entry methods are Boolean equation, truth table, and state diagram. Some software also allows for schematic entry.

The Programmer The **programmer** is connected to the computer and the SPLD is inserted into the programmer socket, which is usually a ZIF socket (zero insertion force socket). The programmer has a software driver program that reads the JEDEC file generated by the compiler software and converts it to instructions for applying required voltages to specified pins on the SPLD in order to program the specified logic design into the SPLD. All SPLD software and programmers, regardless of the manufacturer, conform to a standard for generation of the JEDEC file established by the Joint Electronic Device Engineering Council.

The Programming Process

The general flow chart in Figure 12–21 shows the sequence for the conventional programming of an SPLD. First, the logic circuit is designed and expressed in terms of Boolean equations, truth tables, logic schematic, or other acceptable form, and the software is loaded into the computer.


Start
Design the logic circuit
Debug
Enter design into computer by creating an input (source) file
Edit
Syntax or other errors?
Yes
No
Compiler processes input file and minimizes logic
Design simulation
Design flaws?
No
Yes
Compiler creates JEDEC file (fuse map)
Download to programmer
Programmer "burns" fuse map into SPLD array
Compiler generates documentation file

▲ **FIGURE 12–21**

Flow chart of an SPLD conventional programming sequence

Entering the Design The logic design is entered into the computer by creating an input or source file. Typically, some preliminary information is entered, such as user's name, company, data, and description of the design. Then, the type of SPLD, the input and output pin numbers, and variable labels are entered. Finally, the logic function(s) is entered in the form of a Boolean equation, truth table, schematic diagram, or the like. Any syntax errors or other errors made in entering data into the input file are indicated and must be corrected. **Syntax** is the prescribed format and symbology used to describe a category of functions.

Running the Software The software compiler processes and translates the input file and minimizes the logic. The logic design is then simulated using a set of hypothetical inputs known as *test vectors*. This process effectively "exercises" the design in software to determine if it works properly before the SPLD is actually programmed. If any design flaws are discovered during software simulation, the design must be debugged and modified to correct the flaw.

Once the design is finalized, the compiler generates a **documentation file**, which includes the final logic equations, the JEDEC file, and if desired a pinout diagram of the SPLD. The documentation file essentially provides a "hard copy" of the final design.

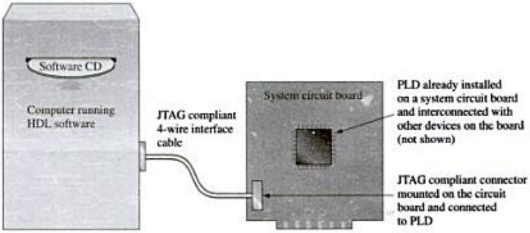
Programming the Device When the design is finalized, the compiler creates a fuse map (JEDEC file) and downloads it to the programmer. The fuse map tells the programmer which fuse links to blow and which to leave intact or, in the case of a GAL, which E^2CMOS cells to turn *on* or *off*. The programmer uses its own software driver program to pattern the SPLD according to the fuse map.

In-System Programmable (ISP) SPLDs

Some SPLDs have ISP capability that allows them to be programmed after they are installed on a circuit board using a standard 4-wire interface specified by IEEE Std. 1149.1, called the **JTAG** (Joint Test Action Group) standard. A JTAG cable is connected from the parallel port of the computer running the programming software to a socket on the PC board that is connected to the special JTAG pins of the SPLD. This configuration is shown in Figure 12–22.

▶ FIGURE 12–22

Typical configuration for in-system programming of a PLD


Software CD
Computer running HDL software
JTAG compliant 4-wire interface cable
System circuit board
PLD already installed on a system circuit board and interconnected with other devices on the board (not shown)
JTAG compliant connector mounted on the circuit board and connected to PLD

In-system programmable devices provide an excellent way to change a circuit design or upgrade a system after it is already in use in the field. An upgrade disk can be used to reprogram a PLD directly on the circuit board via computer or by modem.

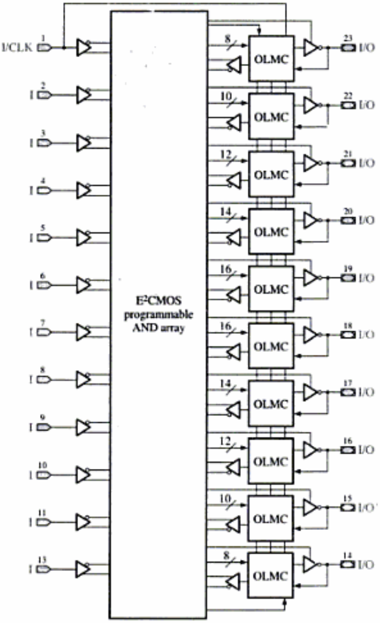
The signals for the JTAG interface standard are as follows:

- TDI—Test Data In. Data, instructions, and addresses are sent to the PLD on this line.
- TDO—Test Data Out. Data are returned from the PLD on this line and compared against expected data.
- TCK—Test Clock. This signal provides timing for the in-system programming process.
- TMS—Test Mode Select. The TMS line controls the transfer of data, instructions, and addresses.

Figure 12–23 shows block diagrams for a conventionally programmable GAL and an ISP GAL. Notice that the only difference is the programming logic with the four JTAG interface lines.

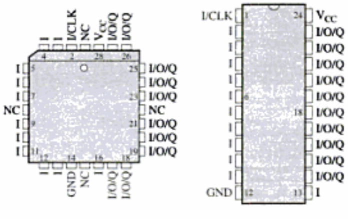
Block Diagram of the GAL22V10

The GAL22V10 contains twelve dedicated inputs and ten input/outputs (I/Os), as shown in the block diagram of Figure 12–24. This device is available in either a 28-pin PLCC (plastic chip carrier) or a 24-pin DIP (dual in-line package), as shown in Figure 12–25. This device is also available in a low-voltage (3.3 V) version, the GAL22LV10, and in in-system programmable versions, the ISP GAL22V10 and the ISP GAL22LV10.


I/CLK 1
I 2
I 3
I 4
I 5
I 6
I 7
I 8
I 9
I 10
I 11
I 13
E²CMOS programmable AND array
8 10 12 14 16 16 14 12 10 8
OLMC
23 I/O
22 I/O
21 I/O
20 I/O
19 I/O
18 I/O
17 I/O
16 I/O
15 I/O
14 I/O

▲ **FIGURE 12–24**

GAL22V10 block diagram


I I I/CLK NC V_CC I/O/Q I/O/Q
4 2 28 26
I 5
I
I 7
NC
I 9
I
I 11
25 I/O/Q
I/O/Q
23 I/O/Q
NC
21 I/O/Q
I/O/Q
19 I/O/Q
12 14 16 18
I I GND NC I I/O/Q I/O/Q
I/CLK 1
I
I
I
I
I 6
I
I
I
I
I
GND 12
24 V_CC
I/O/Q
I/O/Q
I/O/Q
I/O/Q
I/O/Q
18 I/O/Q
I/O/Q
I/O/Q
I/O/Q
I/O/Q
13 I

(a) PLCC (b) DIP

▲ **FIGURE 12–25**

GAL22V10 package diagrams

The Output Logic Macrocells (OLMCs)

As stated in the discussion of GALs in Section 12–4, an **OLMC** contains programmable logic circuits that can be configured either for a combinational output or input or for a registered output. In the registered mode (discussed below), the output comes from a flip-flop.

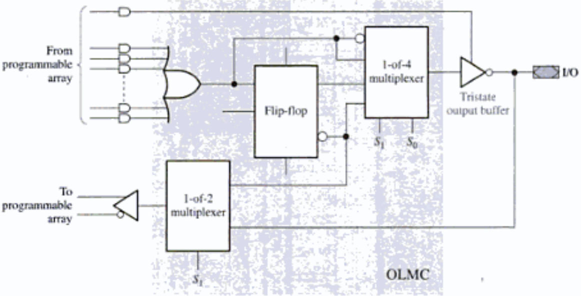
As indicated by the notation in the block diagram of Figure 12–24, of the ten available GAL22V10 OLMCs, two have eight product terms (lines from the AND array to the OR gate), two have ten product terms, two have twelve product terms, two have fourteen product terms, and two have sixteen product terms. Each OLMC can be programmed for either an active-HIGH or an active-LOW output. Also, each OLMC can be programmed as an input.

Logic Diagram A basic logic diagram for the GAL22V10 OLMC is shown in Figure 12–26. The inputs from the AND gates to the OR gate vary from ten to sixteen, as mentioned. The logic inside the shaded box consists of a flip-flop and two multiplexers.

The 1-of-4 multiplexer connects one of its four input lines to the tristate output buffer based on the states of two select inputs, S_0 and S_1. The inputs to the 1-of-4 multiplexer are the OR gate output, the complement of the OR gate output, the flip-flop output, and the complement of the flip-flop output. The 1-of-2 multiplexer connects either the output of the tristate buffer or the flip-flop back through a buffer to the AND array based on the state of S_1. The select bits, S_0 and S_1, for each OLMC are programmed into a dedicated group of cells in the array.

The four OLMC configurations are:

- Combinational mode with active-LOW output
- Combinational mode with active-HIGH output
- Registered mode with active-LOW output
- Registered mode with active-HIGH output



▲ FIGURE 12–26

The GAL22V10 OLMC

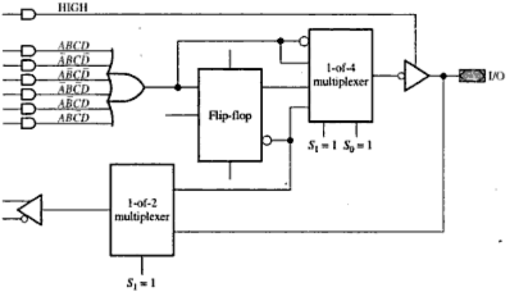
The Combinational Mode

The modes in the GAL22V10 are determined by the S_0 and S_1 bits, which are controlled by programming. For the combinational mode, $S_1S_0 = 10$ or $S_1S_0 = 11$.

When $S_0 = 0$ and $S_1 = 1$, the multiplexer selects the OR gate output. The *effective* logic paths of the OLMC are shown in Figure 12–27(a) where the output polarity is active-LOW because of the inversion of the tristate output buffer. When $S_0 = 1$ and $S_1 = 1$, the complement of the OR gate output is selected, and the *effective* logic paths of the OLMC are as shown in part (b). The output is active-HIGH because of the double inversion (complement of

EXAMPLE 12–5

Determine the output expression of an OLMC in a GAL22V10 for the product terms from the AND array and the select bits as shown in Figure 12–30.



▲ **FIGURE 12–30**

Solution The OLMC is configured for an active-HIGH combinational output; therefore, the SOP expression is

$$X = ABCD + \bar{A}BC\bar{D} + A\bar{B}C\bar{D} + \bar{A}B\bar{C}D + A\bar{B}CD + AB\bar{C}D$$

Supplementary Problem Write the SOP expression for the output if $S_0 = 0$ and $S_1 = 1$.

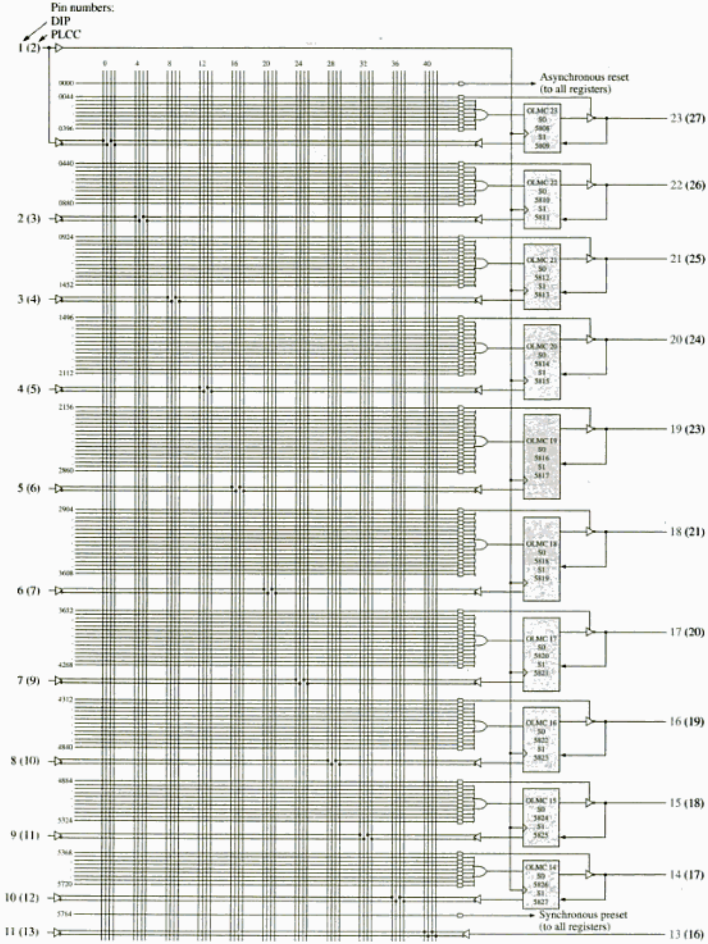
The GAL22V10 Array

The diagram for the GAL22V10 in Figure 12–31 is similar to the block diagram, shown in Figure 12–24, except that details of the programmable AND array are shown.

The GAL22V10 programmable array is organized as 22 input lines and complements crossing 132 product term lines. There is a programmable E^{2}CMOS cell at each of the 5808 intersections. Each cell can be programmed in the *on* state to connect an input variable or its complement to a product term line.

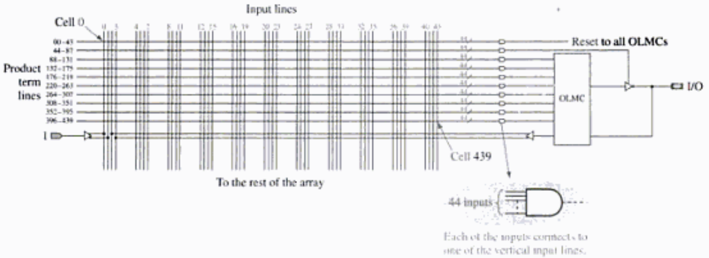
A single line represents the product term line in the diagram of Figure 12–31. However, in the GAL22V10 there are actually 44 lines to each AND gate. So, each product term line consists of 44 AND gate inputs, one for each input line and its complement. The numbers shown for each product term line are the beginning cell numbers for the 44 cells associated with that line. The number in each OLMC for S_0 and S_1 are the cell numbers in the array into which the bits are programmed. These twenty special cells (5808 through 5827) are not shown in the array diagram.

Figure 12–32 shows details of the programmable array for the first OLMC section. The OLMC has a capacity for eight product terms that can be used in an SOP function. One product term is reserved for the tristate control, and one product term is used for a reset function in the registered mode for all of the OLMCs. The latter product term is called a *global* function because it is common to all OLMCs.


Pin numbers:
DIP
PLCC
1 (2)
2 (3)
3 (4)
4 (5)
5 (6)
6 (7)
7 (9)
8 (10)
9 (11)
10 (12)
11 (13)
0
4
8
12
16
20
24
28
32
36
40
0000
0044
0396
0440
0880
0924
1452
1496
2112
2156
2860
2904
3608
3652
4268
4312
4840
4884
5324
5368
5720
5764
Asynchronous reset (to all registers)
OLMC 23 S0 5808 S1 5809
OLMC 22 S0 5810 S1 5811
OLMC 21 S0 5812 S1 5813
OLMC 20 S0 5814 S1 5815
OLMC 19 S0 5816 S1 5817
OLMC 18 S0 5818 S1 5819
OLMC 17 S0 5820 S1 5821
OLMC 16 S0 5822 S1 5823
OLMC 15 S0 5824 S1 5825
OLMC 14 S0 5826 S1 5827
23 (27)
22 (26)
21 (25)
20 (24)
19 (23)
18 (21)
17 (20)
16 (19)
15 (18)
14 (17)
Synchronous preset (to all registers)
13 (16)

▲ **FIGURE 12–31**

GAL22V10 array diagram


Input lines
Cell 0
Product term lines
Reset to all OLMCs
OLMC
I/O
I
Cell 439
To the rest of the array
44 inputs
Each of the inputs connects to one of the vertical input lines.

▲ **FIGURE 12–32**

Organization of the programmable array showing one OLMC portion

Implementing an SOP Function

As mentioned, a cell is programmed to the *on* state where a variable or its complement is part of a product term. Programming a cell to the *on* state simply connects an input line to a product term line. Each AND gate in the array produces one product term. When all of the input variables that are connected to an AND gate are HIGH, the output of the AND gate is HIGH.

The product terms from all of the AND gates for a given OLMC are ORed to form the sum-of-products function. The OLMC select bits are programmed to route the OR gate output, either complemented or uncomplemented, to the output pin.

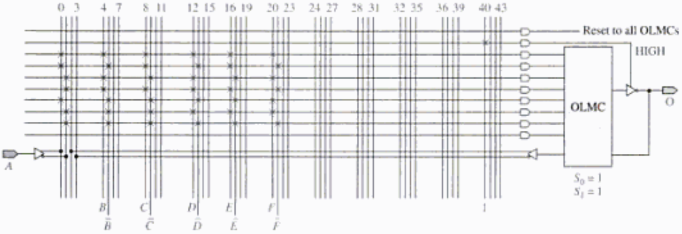
With the GAL22V10, up to ten separate SOP functions can be programmed, the largest of which can contain sixteen product terms. If an SOP expression with more than sixteen terms is required, two OLMCs are necessary. The OR gate output of one OLMC is fed back to the array and connected into the other OLMC by programming.

EXAMPLE 12–6

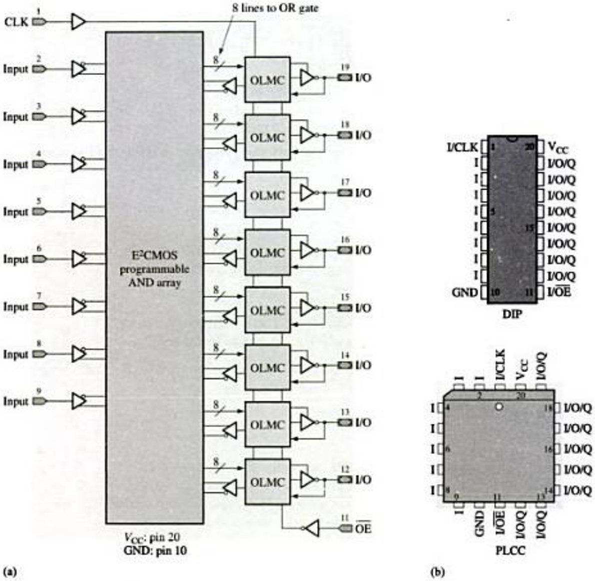
Show how the following 6-variable SOP function is implemented with the GAL22V10.

$$X = ABCDEF + A\bar{B}C\bar{D}E\bar{F} + \bar{A}B\bar{C}D\bar{E}F + \bar{A}BCDE\bar{F} + A\bar{B}\bar{C}\bar{D}\bar{E}F + \bar{A}\bar{B}\bar{C}DEF + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}$$

Solution The programmed portion of the array with the associated OLMC is shown in Figure 12–33. The rest of the array remains unused in this case. The Xs represent E^2CMOS cells that are programmed to the *on* state.


0 3
4 7
8 11
12 15
16 19
20 23
24 27
28 31
32 35
36 39
40 43
Reset to all OLMCs
HIGH
OLMC
O
A
$S_0 = 1$
$S_1 = 1$
B
$\bar{B}$
C
$\bar{C}$
D
$\bar{D}$
E
$\bar{E}$
F
$\bar{F}$
1

▲ **FIGURE 12–33**


8 lines to OR gate
CLK
1
Input
2
3
4
5
6
7
8
9
E²CMOS programmable AND array
8
OLMC
19
18
17
16
15
14
13
12
I/O
11
OE
V_{CC}: pin 20
GND: pin 10
(a)
I/CLK
I
GND
V_{CC}
I/O/Q
I/OE
1
5
10
20
15
11
DIP
I/CLK
V_{CC}
2
20
4
6
8
9
11
13
14
16
18
GND
PLCC
(b)

▲ **FIGURE 12–35**

GAL16V8 block diagram and packaging

► **TABLE 12–1**

GAL16V8 SIMPLE MODE	GAL16V8 COMPLEX MODE	GAL16V8 REGISTERED MODE
PAL10L8	PAL16L8	PAL16R8
PAL12L6	PAL16H6	PAL16R6
PAL14L4	PAL16P8	PAL16R4
PAL16L2		PAL16RP8
PAL10H8		PAL16RP6
PAL12H6		PAL16RP4
PAL14H4		
PAL16H2		
PAL10P8		
PAL12P6		
PAL14P4		
PAL16P2		

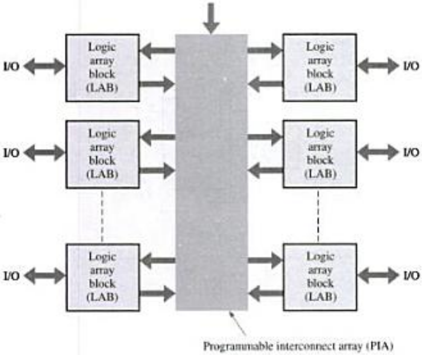
12-8 INTRODUCTION TO CPLDs

So far, you have learned about SPLDs, particularly the PAL and the GAL. In this section, another class of programmable logic devices, called CPLDs, is introduced. A CPLD (Complex Programmable Logic Device) is a logic device that consists of multiple SPLDs interconnected on a single chip. CPLDs can be used to implement large logic functions, including shift registers.

After completing this section, you should be able to

- Describe a basic CPLD
- Explain the purpose of the logic array block (LAB)
- Explain the purpose of the programmable interconnect array (PIA)
- Discuss a basic macrocell
- Describe the MAX 7000 CPLDs

A **CPLD** basically consists of multiple groups of PAL/GAL-like arrays with programmable interconnections, as shown in the general block diagram in Figure 12–39. Each PAL/GAL group is called a **logic array block (LAB),** *function block*, or some similar term depending on the particular device. Each logic array block contains several PAL/GAL-like arrays called macrocells. Each LAB can be interconnected with other LABs or to other I/Os (input/output) using the programmable interconnect array (PIA) to form large complex logic functions. Like the PAL or GAL, the CPLD is based on a sum-of-products (SOP) architecture.



▲ **FIGURE 12–39**

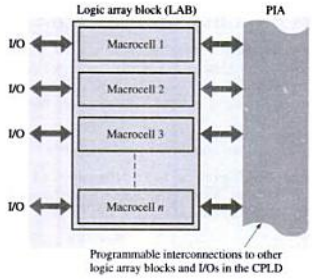
Basic diagram of a CPLD

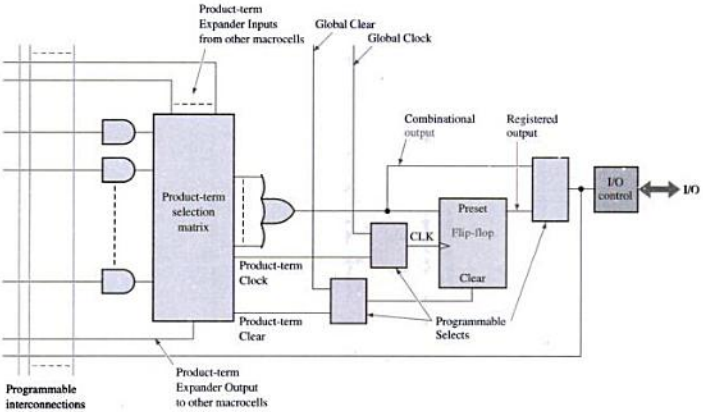
Macrocells

Each logic array block in a CPLD contains several macrocells, as shown in Figure 12–40. CPLD architecture varies from one manufacturer to another, but generally there are from 32 to several hundred macrocells in one LAB. A typical **macrocell** has an AND array, a product-term select matrix, an OR gate, and a programmable register section. A simplified diagram of a typical CPLD macrocell is shown in Figure 12–41. Notice that the logic is similar to the OLMC logic in a PAL/GAL array.

▶ **FIGURE 12-40**

Basic logic array block in a CPLD


Logic array block (LAB)
PIA
I/O
I/O
I/O
I/O
Macrocell 1
Macrocell 2
Macrocell 3
Macrocell *n*
Programmable interconnections to other logic array blocks and I/Os in the CPLD


Product-term Expander Inputs from other macrocells
Global Clear
Global Clock
Combinational output
Registered output
Product-term selection matrix
Preset
CLK
Flip-flop
Clear
I/O control
I/O
Product-term Clock
Product-term Clear
Programmable Selects
Product-term Expander Output to other macrocells
Programmable interconnections

▲ **FIGURE 12-41**

Basic CPLD macrocell

Each macrocell has a fixed number of AND gates that feed into a product-term selection matrix, where product terms can be selected and applied to an OR gate. Additionally, product-term expander inputs from other macrocells allow more product terms to be selected in addition to those from the macrocell AND array. Also, a product-term expander output provides any selected product term to other macrocells in the LAB or in other LABs via the programmable interconnect array (PIA).

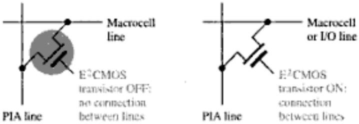
The OR gate provides an SOP output through programmable select blocks to the I/O or to a flip-flop. In this particular implementation, there are three programmable selects, which are essentially data selectors (multiplexers). One programmable select block provides either a global clock or a product term to be used as the clock input for the flip-flop. A second programmable select block provides either a global clear or a product-term clear to the flip-flop. The third programmable select block routes either the output of the OR gate or the output of the flip-flop to the I/O. The OR gate provides an SOP (combinational) output, and the flip-flop provides a registered output.

In CPLDs, the term *registered* is used in reference to the flop-flop and its associated circuits. The flip-flops in a CPLD can be used for implementing shift register or counter logic. Other flip-flops can be implemented using gate logic as was discussed in Chapter 8. The determination of whether to use the macrocell flip-flop or a gate-implemented flip-flop is made by the programming software.

Programmable Interconnect Array (PIA)

The **programmable interconnect array (PIA)** consists of conductors that run throughout the CPLD chip and to which connections from the macrocells in each LAB can be made. By using the PIA, any macrocell can be connected to other macrocells within the same LAB, to macrocells in other LABS in the device, or to other I/Os.

Connections to the AND gates or other elements in a macrocell are accomplished by connecting a line in the programmable interconnect array to an AND gate input or other macrocell line. Most CPLDs use E^{2}CMOS technology to make the connections. In E^{2}CMOS, a transistor between two lines is programmed to the *on* state to form a connection and in the *off* state for no connection. The basic idea of this connection method is illustrated in Figure 12–42.


Macrocell line
E²CMOS transistor OFF: no connection between lines
PIA line
Macrocell or I/O line
E²CMOS transistor ON: connection between lines
PIA line

▲ **FIGURE 12–42**

Basic E^{2}CMOS interconnection technolog

A Specific CPLD

Several companies, including Altera, Xilinx, Lattice, Cypress, and others provide CPLDs. Each company has its own approach to CPLD architecture, but all have one thing in common—they are based on PAL/GAL SOP logic arrays. Since we cannot cover all of the devices that are available, a specific CPLD family is introduced as a representative example. Altera's MAX 7000 family of CPLDs is used because it is one of the most popular on the market. Altera also has other CPLD families that differ somewhat in architecture. Some CPLDs must be programmed in a special fixture; others, such as the MAX 7000S, are in-system programmable (ISP). ISP means that a CPLD can be programmed while mounted on a system printed circuit board.

The MAX 7000 family of CPLDs includes several variations ranging from 2 to 32 logic array blocks each with 16 macrocells. These CPLDs can be programmed and erased up to 100 times. They are supported by the manufacturer's development systems, which are packages that include schematic entry, text entry, and waveform design entry as well as compilation software, simulation and timing analysis, and device programming.

Figure 12–43 shows the general block diagram for the MAX 7000 family. The logic array blocks (LABs) each consist of sixteen macrocells, and the LABs are linked together by the programmable interconnect array (PIA). The PIA is fed by all input/outputs (I/Os) and the macrocells. The arrows with slashes indicate numbers of parallel lines. Each macrocell has the following inputs:

12–9 INTRODUCTION TO FPGAs

In Section 12–8, the CPLD was introduced. Now, another major class of programmable logic devices, called the field programmable gate array (FPGA), is introduced. Recall that CPLDs consist essentially of multiple PAL/GAL-type logic blocks that are linked by programmable interconnections. CPLDs are based on SOP (sum-of-product) logic. Although there are some similarities, FPGAs are distinctly different from CPLDs in terms of their architecture and generally offer a much higher logic capacity.

After completing this section, you should be able to

- Describe a basic FPGA
- Describe the FPGA logic block
- Discuss the FPGA logic element
- Explain how a look-up-table (LUT) works

Although there are many variations of the basic architecture, the **FPGA** basically consists of an array of logic blocks with programmable row and column interconnecting channels surrounded by programmable I/O blocks, as shown in Figure 12–44. Many FPGA architectures are based on a type of memory called an **LUT** (look-up table) rather than on SOP AND/OR arrays as CPLDs are. Another approach found on some FPGAs is the use of multiplexers to generate logic functions, which was discussed earlier in this chapter.

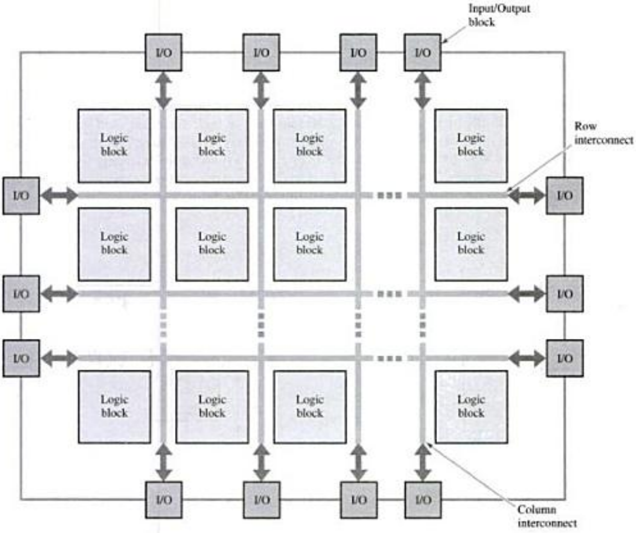


FIGURE 12–44

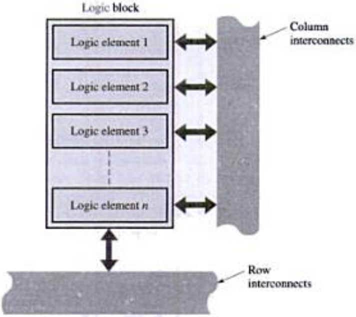
Basic block diagram of an FPGA

The Logic Block

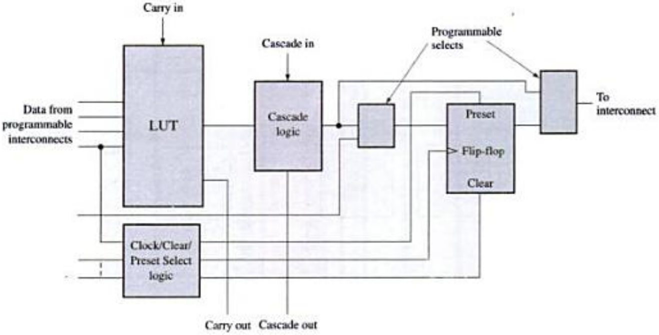
Each logic block in a generic FPGA contains several logic elements, as shown in Figure 12–45. As mentioned, FPGA architecture varies from one manufacturer to another, but generally there can be well over ten thousand logic elements in a single chip.

▶ **FIGURE 12–45**

Basic logic block in an FPGA

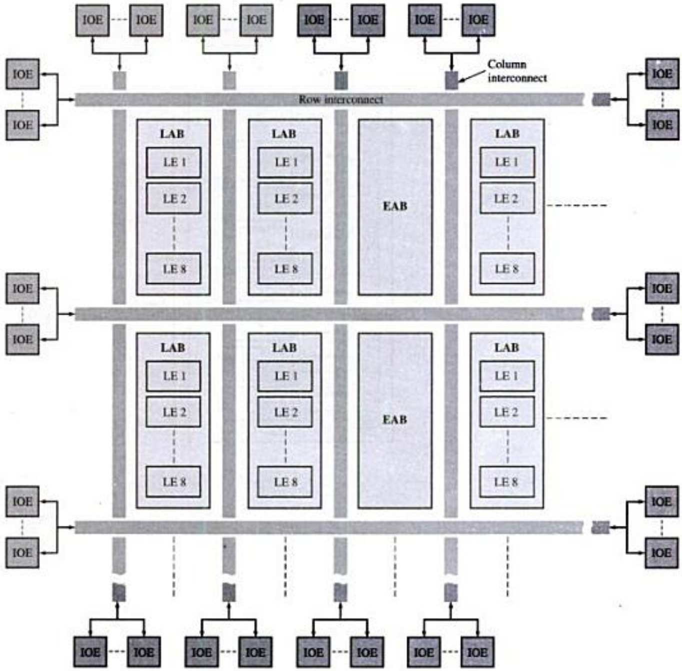

Logic block
Logic element 1
Logic element 2
Logic element 3
Logic element *n*
Column interconnects
Row interconnects

The Logic Element A simplified diagram of a typical FPGA **logic element** is shown in Figure 12–46. Notice that it contains an LUT, associated logic, and a flip-flop. In this case, each logic element contains a 4-input LUT that can be programmed as a logic function generator. It can be used to produce SOP functions or logic functions such as adders and comparators. When configured as an adder, the carry in and carry out allow for adder expansion. Using the cascade logic, an LUT can be expanded by cascading with LUTs in other logic elements. The programmable selects let you choose either combinational functions from the LUT output or registered functions from the flip-flop output.


Carry in
Cascade in
Programmable selects
Data from programmable interconnects
LUT
Cascade logic
Preset
Flip-flop
Clear
To interconnect
Clock/Clear/ Preset Select logic
Carry out
Cascade out

▲ **FIGURE 12–46**

A simplified typical FPGA logic element


IOE
Column interconnect
Row interconnect
LAB
LE 1
LE 2
LE 8
EAB

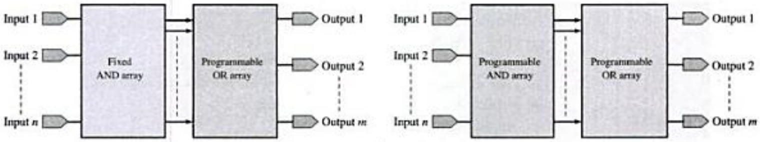
▲ **FIGURE 12–50**

Simplified FLEX 10K block diagram showing the basic architecture

specified logic functions. This particular LUT can generate any 4-variable logic function because it has four inputs. The basic logic element is shown in Figure 12–52. Notice that in addition to the LUT, there is the carry chain and the cascade chain. Also, notice that the LE can be connected to the local interconnect or the main interconnects (row and column) with the select logic.

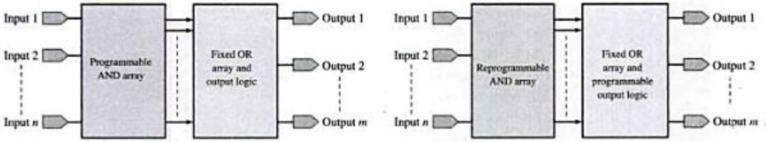
SUMMARY

- Basic types of SPLDs (simple programmable logic devices) are PAL (programmable array logic), GAL (generic array logic), PLA (programmable logic array), and PROM (programmable read-only memory).
- Classifications of SPLDs are PROM, PLA, PAL, and GAL, as shown in Figure 12–53.


Input 1
Input 2
Input n
Fixed AND array
Programmable OR array
Output 1
Output 2
Output m
Input 1
Input 2
Input n
Programmable AND array
Programmable OR array
Output 1
Output 2
Output m

(a) PROM

(b) PLA


Input 1
Input 2
Input n
Programmable AND array
Fixed OR array and output logic
Output 1
Output 2
Output m
Input 1
Input 2
Input n
Reprogrammable AND array
Fixed OR array and programmable output logic
Output 1
Output 2
Output m

(c) PAL

(d) GAL

▲ **FIGURE 12–53**

- PALs are one-time programmable (OTP).
- PALs can produce any sum-of-products logic function limited only by the number of variables.
- A PAL consists of a programmable AND array and a fixed OR array with programmable output logic.
- The generic array logic (GAL) consists of a reprogrammable AND array, a fixed OR array, and programmable output logic.
- A conventionally programmable SPLD requires a computer, HDL compiler software, and a programming fixture (programmer).
- An in-system programmable (ISP) SPLD requires a computer, HDL compiler software, and a JTAG interface cable. Also, the circuit board's ISP SPLD must be JTAG compatible.
- GALs can emulate most types of PALs.
- An OLMC (output logic macrocell) contains programmable logic circuits that can be configured either for a combinational output or input or for a registered output.
- An OLMC (Output Logic Macrocell) can be configured for either of two modes: the combinational mode or the registered mode.
- In the OLMC registered mode, the output comes from a D flip-flop.
- A CPLD is a complex programmable logic device.
- A CPLD consists of logic array blocks with programmable interconnections.

SELF-TEST

Answers are at the end of the chapter.

1. A CPLD is a

(a) CMOS programmable logic device **(b)** Capacitive programmable logic device
(c) Complex programmable logic device **(d)** Complementary process latching device

2. VHDL is a

(a) logic device **(b)** PLD programming language
(c) computer language **(d)** very high density logic

3. The types of SPLDs do not include

(a) GAL **(b)** PROM **(c)** RAM **(d)** PAL

4. A GAL has

(a) a reprogrammable AND array, a fixed OR array, and programmable output logic
(b) a fixed AND array and a programmable OR array
(c) one-time programmable AND and OR arrays
(d) reprogrammable AND and OR arrays

5. An SPLD that has a programmable AND array and a fixed OR array is a

(a) PROM **(b)** PLA **(c)** PAL **(d)** GAL

6. A connection between a row and column in a PAL array is made by

(a) blowing a fusible link
(b) leaving a fusible link intact
(c) connecting an input variable to the input line
(d) connecting an input variable to the product term line

7. The device number PAL14H4 indicates

(a) a PAL with fourteen active-HIGH outputs and four inputs
(b) a PAL that implements fourteen AND gates and four OR gates
(c) a PAL with fourteen inputs and four active-HIGH outputs
(d) who the manufacturer is

8. A GAL is different from a PAL because

(a) a GAL has more inputs and outputs
(b) a GAL is implemented with a different technology
(c) a GAL can replace several different PALs
(d) a GAL can be reprogrammed and a PAL cannot
(e) all of the above answers
(f) all except answer (a)
(g) all except answer (c)

9. The reprogrammable cells in a GAL array are

(a) TTL **(b)** E^2CMOS **(c)** ECL **(d)** bipolar fuses

10. OLMC is an acronym for

(a) Output Logic Main Cell
(b) Optimum Logic Multiple Channel
(c) Output Logic Macrocell
(d) Odd-parity Logic Master Check

11. Two ways in which a GAL output can be configured are

(a) combinational and I/O
(b) simple and complex
(c) fixed and variable
(d) combinational and registered

12. The device number GAL22V10 means that
    - **(a)** the device has 22 dedicated inputs and 10 dedicated outputs
    - **(b)** the device has 22 inputs including dedicated inputs and I/Os and 10 outputs either dedicated or I/Os
    - **(c)** the device has a variable number of inputs from a maximum of 22 to a minimum of 10
13. To conventionally program an SPLD, you need a
    - **(a)** special fixture
    - **(b)** special fixture and a master PLD that has been preprogrammed at the factory
    - **(c)** computer and a programmer
    - **(d)** computer, a programmer, and HDL software
    - **(e)** computer, a programmer, and BASIC software
14. ISP stands for
    - **(a)** In-System Programmable
    - **(b)** Integrated System Program
    - **(c)** Integrated Silicon Programmer
15. The GAL22V10 has
    - **(a)** 10 inputs and 22 outputs
    - **(b)** 22 dedicated inputs and 10 outputs
    - **(c)** 12 dedicated inputs and 10 dedicated outputs
    - **(d)** 12 dedicated inputs and 10 outputs, any of which can be an input
16. The GAL22LV10 operates on a dc supply voltage of

    (a) 5 V **(b)** 10 V **(c)** 3.3 V **(d)** 1.2 V
17. OLMC stands for
    - **(a)** output logic modular circuit
    - **(b)** output logic macrocell
    - **(c)** output latch memory cell
    - **(d)** overall logic matrix circuit
18. The three states of a tristate output buffer are
    - **(a)** HIGH, LOW, high impedance
    - **(b)** HIGH, LOW, ground
    - **(c)** HIGH, LOW, in between
    - **(d)** Right, left, center
19. The OLMC of the GAL22V10 contains
    - **(a)** one OR gate, one flip-flop, two multiplexers
    - **(b)** one OR gate, one flip-flip, one multiplexer
    - **(c)** one AND gate, one latch, two multiplexers
    - **(d)** one OR gate, one flip-flop, two decoders
20. The GAL16V8 has
    - **(a)** 16 dedicated inputs and 8 outputs
    - **(b)** 8 dedicated inputs and 8 inputs/outputs
    - **(c)** 8 inputs and 16 outputs
    - **(d)** 16 input/outputs and 8 outputs
21. A typical OLMC consists of
    - **(a)** gates, multiplexers, and a flip-flop
    - **(b)** gates and a shift register
    - **(c)** a Gray code counter
    - **(d)** a fixed logic array
22. A CPLD is a
    - **(a)** CMOS PLD
    - **(b)** complex PLD
    - **(c)** complementary PLD
    - **(d)** capacitive PLD
23. A CPLD contains
    - **(a)** shift registers
    - **(b)** programmable interconnections
    - **(c)** logic arrays
    - **(d)** answers (b) and (c)

24. FPGA stands for

(a) fast propagation gate array
(b) field presettable gate application
(c) field programmable gate array
(d) file programmable gate array

PROBLEMS

Answers to odd-numbered problems are at the end of the book.

SECTION 12–1 Introduction to Programmable Logic Devices (PLDs)

1. Which of the following acronyms do not describe a PLD?
PAL, GAL, SPLD, ABEL, CPLD, CUPL, EPLD, EEPLD, FPGA

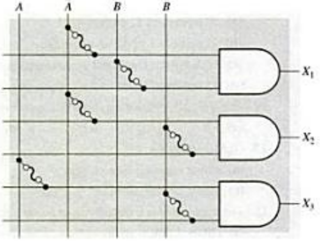
2. What do each of the following stand for?

(a) SPLD **(b)** CPLD **(c)** HDL **(d)** FPGA **(e)** GAL

SECTION 12–2 Simple Programmable Logic Devices (SPLDs)

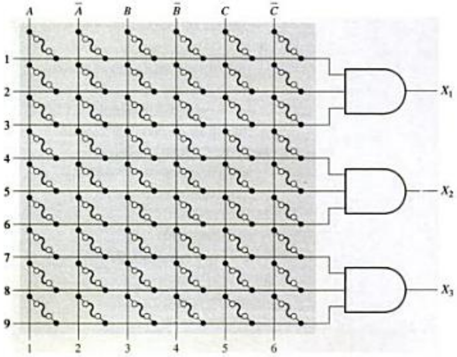
3. In the simple programmed AND array with fusible links in Figure 12–54, determine the Boolean output expressions.

FIGURE 12–54



4. Determine by row and column number which fusible links must be blown in the programmable AND array of Figure 12–55 to implement each of the following product terms: $X_1 = \bar{A}BC$, $X_2 = AB\bar{C}$, $X_3 = \bar{A}B\bar{C}$.

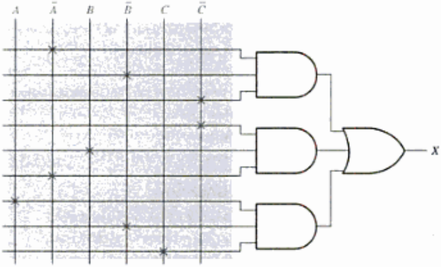
FIGURE 12–55



SECTION 12–3 Programmable Array Logic (PAL)

5. Determine the Boolean output expression for the simple PAL shown in Figure 12–56, where the Xs represent intact fusible links.

► FIGURE 12–56

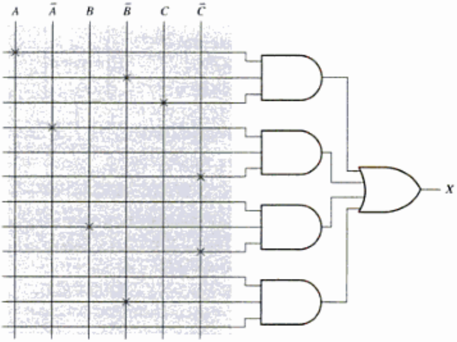


6. Show how the PAL-type array in Figure 12–56 should be programmed in order to implement each of the following SOP expressions. Use an X to indicate an intact fuse. Simplify the expressions, if necessary, to fit the PAL-type array shown.
   (a) $Y = A\bar{B}C + \bar{A}B\bar{C} + ABC$
   (b) $Y = A\bar{B}C + \bar{A}\,\bar{B}C + A\bar{B}\bar{C} + \bar{A}BC$
7. Interpret each of the PAL device numbers.
   (a) PAL16L2 (b) PAL12H6 (c) PAL10P8 (d) PAL16R6
8. Explain how a programmed polarity output in a PAL works.

SECTION 12–4 Basic Concepts of GAL

9. Determine the output expression for the GAL type array shown in Figure 12–57. The Xs represent an *on* cell.
10. Show how the GAL type array in Figure 12–57 should be reprogrammed in order to implement each of the following expressions. Use Xs to indicate *on* cells. Modify the expressions, if necessary.
    (a) $X = A\bar{B}C + \bar{A}B\bar{C} + A\bar{B} + BC$ (b) $X = (A + \bar{B} + \bar{C})(\bar{A} + B)$

► FIGURE 12–57



SECTION 12–7 The GAL16V8

28. Describe a GAL16V8 in terms of
    (a) the number of dedicated inputs
    (b) the number of I/Os
    (c) the number of dedicated inputs plus the number of I/Os
29. If there are ten inputs to a GAL16V8, what is the maximum number of available outputs?
30. What is the largest single SOP expression that can be implemented with a GAL16V8 in terms of the number of variables and the number of product terms? Assume that only one output is used.
31. PAL emulation with a GAL16V8 can be achieved by programming the device in any one of three modes. What are these modes?
32. Determine the GAL16V8 mode configuration for each combination of "cell" values.
    (a) $\overline{SYN} = 0, ACO = 1, XOR = 1$
    (b) $\overline{SYN} = 1, ACO = 0, XOR = 0$
33. Besides the flip-flop and logic gates, what other type of device is used in an OLMC?
34. Explain what a *global* cell is.

ANSWERS

SECTION REVIEWS

SECTION 12–1 Introduction to Programmable Logic Devices (PLDs)

1. Programmable logic device
2. Simple PLD
3. Complex PLD
4. Field-programmable gate array
5. A CPLD has a much higher logic capacity than an SPLD.
6. Schematic entry, text-based entry
7. Hardware description language
8. VHDL and Verilog HDL
9. VHDL

SECTION 12–2 Simple Programmable Logic Devices (SPLDs)

1. Four types of SPLDs are PROM, PLA, PAL, and GAL.
2. In a PLA, both arrays are programmable. In a PAL, only the AND array is programmable.
3. A PAL is one-time programmable; a GAL is reprogrammable and has programmable output sections.

SECTION 12–3 Programmable Array Logic (PAL)

1. A PAL is an SPLD with a programmable AND array and a fixed OR array.
2. The fusible link is the programmable element in an array.
3. Any combinational logic can be implemented with a PAL within the limitations of available inputs and outputs.
4. Three types of PAL output logic are combinational output, combinational I/O, and programmable polarity output.
5. A PAL12H6 is a PAL that has 12 inputs, including I/Os, and 6 active-HIGH outputs.

ANSWERS TO ODD-NUMBERED PROBLEMS

Chapter 1

1. Digital can be transmitted and stored more efficiently and reliably.

3. **(a)** 11010001 **(b)** 000101010

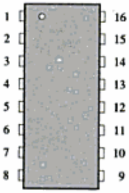
5. **(a)** 550 ns **(b)** 600 ns
(c) 2.7 μs **(d)** 10 V

7. 250 Hz **9.** 50%

11. 8 μs; 1 μs **13.** AND gate

15. An IC is an electronic circuit with all components integrated on a single silicon chip.

17.



Chapter 2

1. **(a)** 1 **(b)** 100 **(c)** 100,000

3. **(a)** 400; 70; 1 **(b)** 9000; 300; 50; 6
(c) 100,000; 20,000; 5000; 0; 0; 0

5. **(a)** 3 **(b)** 4 **(c)** 7 **(d)** 8 **(e)** 9
(f) 12 **(g)** 11 **(h)** 15

7. **(a)** 51.75 **(b)** 42.25 **(c)** 65.875
(d) 120.625 **(e)** 92.65625 **(f)** 113.0625
(g) 90.625 **(h)** 127.96875

9. **(a)** 5 bits **(b)** 6 bits **(c)** 6 bits
(d) 7 bits **(e)** 7 bits **(f)** 7 bits
(g) 8 bits **(h)** 8 bits

11. **(a)** 1010 **(b)** 10001 **(c)** 11000
(d) 110000 **(e)** 111101 **(f)** 1011101
(g) 1111101 **(h)** 10111010

13. **(a)** 1111 **(b)** 10101 **(c)** 11100
(d) 100010 **(e)** 101000 **(f)** 111011
(g) 1000001 **(h)** 1001001

15. **(a)** 100 **(b)** 100 **(c)** 1000
(d) 1101 **(e)** 1110 **(f)** 11000

17. **(a)** 1001 **(b)** 1000 **(c)** 100011
(d) 110110 **(e)** 10101001 **(f)** 10110110

19. **(a)** 010 **(b)** 001 **(c)** 0101
(d) 00101000 **(e)** 0001010 **(f)** 11110

21. **(a)** 00011101 **(b)** 11010101
(c) 01100100 **(d)** 11111011

23. **(a)** 00001100 **(b)** 10111100
(c) 01100101 **(d)** 10000011

25. **(a)** −102 **(b)** +116 **(c)** −64

27. **(a)** 0 10001101 11110000101011000000000
(b) 1 10001010 10000011000000000000000

29. **(a)** 00110000 **(b)** 00011101
(c) 11101011 **(d)** 100111110

31. **(a)** 11000101 **(b)** 11000000

33. 100111001010

35. **(a)** 00111000
(b) 01011001
(c) 101000010100
(d) 010111001000
(e) 0100000100000000
(f) 1111101100010111
(g) 1000101010011101

37. **(a)** 35 **(b)** 146 **(c)** 26 **(d)** 141
(e) 243 **(f)** 235 **(g)** 1474 **(h)** 1792

39. **(a)** 60_{16} **(b)** $10B_{16}$ **(c)** $1BA_{16}$

41. **(a)** 10 **(b)** 23 **(c)** 46 **(d)** 52
(e) 67 **(f)** 367 **(g)** 115 **(h)** 532
(i) 4085

43. **(a)** 001011 **(b)** 101111
(c) 001000001
(d) 011010001
(e) 101100000
(f) 100110101011
(g) 001011010111001
(h) 100101110000000
(i) 001000000010001011

45. **(a)** 00010000 **(b)** 00010011
(c) 00011000 **(d)** 00100001
(e) 00100101 **(f)** 00110110
(g) 01000100 **(h)** 01010111

(i) 01101001 (j) 10011000
(k) 000100100101 (l) 000101010110

47. (a) 000100000100 (b) 000100101000
(c) 000100110010 (d) 000101010000
(e) 000110000110 (f) 001000010000
(g) 001101011001 (h) 010101000111
(i) 0001000001010001

49. (a) 80 (b) 237 (c) 346 (d) 421
(e) 754 (f) 800 (g) 978 (h) 1683
(i) 9018 (j) 6667

51. (a) 00010100 (b) 00010010
(c) 00010111 (d) 00010110
(e) 01010010 (f) 000100001001
(g) 000110010101 (h) 0001001001101001

53. The Gray code makes only one bit change at a time when going from one number in the sequence to the next.

55. (a) 1100 (b) 00011 (c) 10000011110

57. (a) CAN (b) J (c) =
(d) # (e) > (f) B

59. 48 65 6C 6C 6F 2E 20 48 6F 77 20 61 72 65 20 79 6F 75 3F

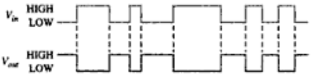
61. (b) is incorrect.

63. (a) 110100100 (b) 000001001
(c) 111111110

65. (a) 1101000 (b) 0000001
(c) 1000010 (d) 0011000

Chapter 3

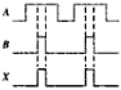
1. See Figure P-1.



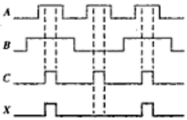
▲ FIGURE P-1

3. See Figure P-2.

► FIGURE P-2

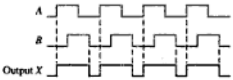


5. See Figure P-3.



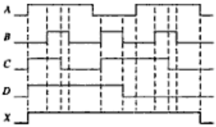
▲ FIGURE P-3

7. See Figure P-4.



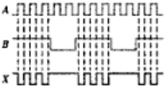
▲ FIGURE P-4

9. See Figure P-5.



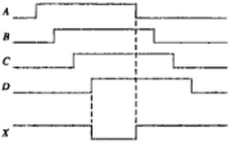
▲ FIGURE P-5

11. See Figure P-6.



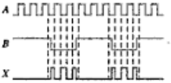
▲ FIGURE P-6

13. See Figure P-7.



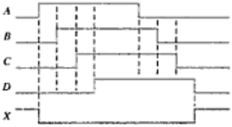
▲ FIGURE P-7

15. See Figure P–8.



▲ FIGURE P–8

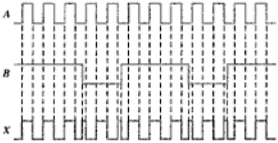
17. See Figure P–9.



▲ FIGURE P–9

19. XOR = $A\overline{B} + \overline{A}B$; OR = $A + B$

21. See Figure P–10.



▲ FIGURE P–10

Chapter 4

1. $X = A + B + C + D$ **3.** $X = \overline{A} + \overline{B} + \overline{C}$

5. **(a)** $AB = 1$ when $A = 1, B = 1$
(b) $A\overline{B}C = 1$ when $A = 1, B = 0, C = 1$
(c) $A + B = 0$ when $A = 0, B = 0$
(d) $\overline{A} + B + \overline{C} = 0$ when $A = 1, B = 0, C = 1$
(e) $\overline{A} + \overline{B} + C = 0$ when $A = 1, B = 1, C = 0$
(f) $\overline{A} + B = 0$ when $A = 1, B = 0$
(g) $A\overline{B}\,\overline{C} = 1$ when $A = 1, B = 0, C = 0$

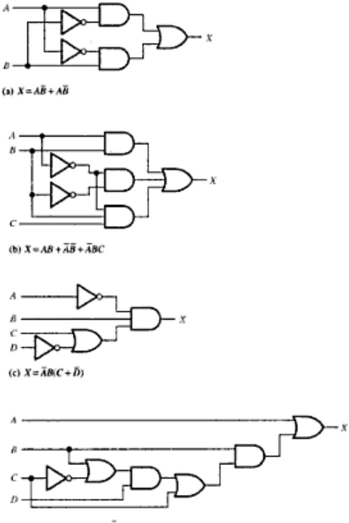
7. **(a)** Commutative **(b)** Commutative
(c) Distributive

9. **(a)** $\overline{A}B$ **(b)** $A + \overline{B}$
(c) $\overline{A}\,\overline{B}\,\overline{C}$ **(d)** $\overline{A} + \overline{B} + \overline{C}$

11. **(a)** $(\overline{A} + \overline{B} + \overline{C})(\overline{E} + \overline{F} + \overline{G})(\overline{H} + \overline{I} + \overline{J})$
$(\overline{K} + \overline{L} + \overline{M})$
(b) $\overline{A}B\overline{C}(\overline{C} + \overline{D}) + BC = \overline{A}B\overline{C} + BC$
(c) $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F}\,\overline{G}\,\overline{H}$

13. **(a)** $X = ABCD$ **(b)** $X = AB + C$
(c) $X = \overline{\overline{A}B}$ **(d)** $X = (A + B)C$

15. See Figure P–11.



(a) $X = A\overline{B} + A\overline{B}$

(b) $X = AB + \overline{A}\overline{B} + \overline{A}BC$

(c) $X = \overline{A}B(C + \overline{D})$

(d) $X = A + B[C + D(B + \overline{C})]$

▲ FIGURE P–11

17. **(a)** A **(b)** AB **(c)** C
(d) A **(e)** $\overline{A}C + \overline{B}C$

19. **(a)** $BD + BE + \overline{D}F$ **(b)** $\overline{A}\,\overline{B}C + \overline{A}\,\overline{B}D$
(c) B **(d)** $AB + CD$
(e) ABC

21. **(a)** $A\overline{B} + AC + BC$ **(b)** $AC + \overline{B}C$
(c) $AB + AC$

23. **(a)** Domain: A, B, C
Standard SOP: $A\overline{B}C + A\overline{B}\,\overline{C} + ABC + \overline{A}BC$

(b) Domain: A, B, C
Standard SOP: $ABC + A\overline{B}C + \overline{A}\,\overline{B}C$

(c) Domain: A, B, C
Standard SOP: $ABC + AB\overline{C} + A\overline{B}C$

25. (a) 101 + 100 + 111 + 011

(b) 111 + 101 + 001

(c) 111 + 110 + 101

27. (a) $(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)$
$(\overline{A} + \overline{B} + C)$

(b) $(A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})$
$(\overline{A} + B + C)(\overline{A} + \overline{B} + C)$

(c) $(A + B + C)(A + B + \overline{C})(A + \overline{B} + C)$
$(A + \overline{B} + \overline{C})(\overline{A} + B + C)$

29. (a) See Table P–1. **(b)** See Table P–2.

▼ **TABLE P–1**

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

▼ **TABLE P–2**

X	Y	Z	Q
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

31. (a) See Table P–3. **(b)** See Table P–4.

▼ **TABLE P–3**

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

▼ **TABLE P–4**

W	X	Y	Z	Q
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

33. **(a)** See Table P–5. **(b)** See Table P–6.

TABLE P–5

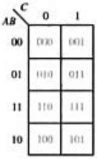
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

TABLE P–6

A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

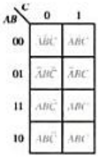
35. See Figure P–12.

FIGURE P–12


C
AB
0
1
00
000
001
01
010
011
11
110
111
10
100
101

37. See Figure P–13.

FIGURE P–13


C
AB
0
1
00
$\overline{A}\overline{B}\overline{C}$
$\overline{A}\overline{B}C$
01
$\overline{A}B\overline{C}$
$\overline{A}BC$
11
$AB\overline{C}$
ABC
10
$A\overline{B}\overline{C}$
$A\overline{B}C$

39. **(a)** No simplification **(b)** AC

(c) $\overline{D}\overline{F} + E\overline{F}$

41. **(a)** $AB + AC$

(b) $A + BC$

(c) $B\overline{C}D + A\overline{C}D + BC\overline{D} + AC\overline{D}$

(d) $A\overline{B} + CD$

43. $\overline{B} + C$ **45.** $\overline{A}\,\overline{B}\,\overline{C}D + C\overline{D} + BC + A\overline{D}$

47. **(a)** $(A + \overline{B} + C + \overline{D})(\overline{A} + B + \overline{C} + D)$
$(\overline{A} + \overline{B} + \overline{C} + \overline{D})$

(b) $(W + \overline{Z})(W + X)(\overline{Y} + \overline{Z})(X + \overline{Y})$

49. $(A + C + D)(A + \overline{B} + C)(\overline{A} + B + \overline{D})$
$(B + \overline{C} + \overline{D})(\overline{A} + \overline{B} + \overline{C} + D)$

51. $X = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} + \overline{A}BC\overline{D}\,\overline{E} + AB\overline{C}\,\overline{D}\,\overline{E} +$
$\overline{A}\,\overline{B}DE + \overline{A}BD\overline{E} + \overline{A}\,\overline{C}DE + AB\overline{C}D$

Chapter 5

1. See Figure P–14.

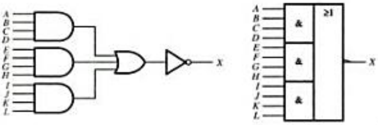

A
B
C
D
E
F
G
H
I
J
K
L
X
&
&
&
≥1
A
B
C
D
E
F
G
H
I
J
K
L
X

FIGURE P–14

3. **(a)** $X = ABB$ **(b)** $X = AB + B$

(c) $X = \overline{A} + B$ **(d)** $X = (A + B) + AB$

(e) $X = \overline{\overline{\overline{A}\,\overline{B}}\,\overline{B}C}$ **(f)** $X = (A + B)(\overline{B} + C)$

5. (a)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(b)

A	B	X
0	0	0
0	1	1
1	0	0
1	1	1

(c)

A	B	X
0	0	1
0	1	1
1	0	0
1	1	1

(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

5. (e)

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

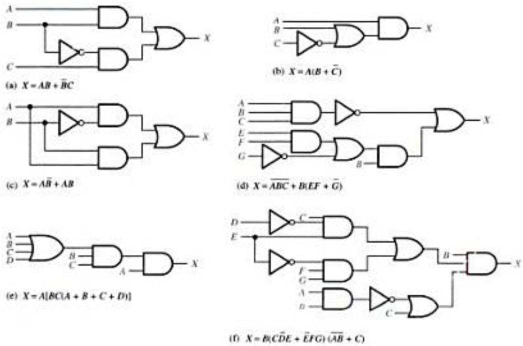
5. (f)

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

7. $X = \overline{A\bar{B} + \bar{A}B} = (\bar{A} + B)(A + \bar{B})$

9. See Figure P–15.

▶ **FIGURE P–15**



(a) $X = AB + \bar{B}C$

(b) $X = A(B + \bar{C})$

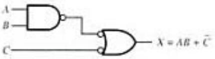
(c) $X = A\bar{B} + AB$

(d) $X = \overline{ABC} + B(EF + \bar{G})$

(e) $X = A[BC(A + B + C + D)]$

(f) $X = B(C\bar{D}E + \bar{E}FG)(\overline{AB} + C)$

11. See Figure P–16.



▲ **FIGURE P–16**

13. $X = AB$

15. **(a)** No simplification

  (b) No simplification

  (c) $X = A$

  (d) $X = \overline{A} + \overline{B} + \overline{C} + EF + \overline{G}$

  (e) $X = ABC$

  (f) $X = BC\overline{D}E + \overline{A}B\overline{E}FG + BC\overline{E}FG$

17. **(a)** $X = AC + AD + BC + BD$

  (b) $X = \overline{A}CD + \overline{B}CD$

  (c) $X = ABD + CD + E$

  (d) $X = \overline{A} + B + D$

  (e) $X = ABD + \overline{C}D + \overline{E}$

  (f) $X = \overline{A}\,\overline{C} + \overline{A}\,\overline{D} + \overline{B}\,\overline{C} + \overline{B}\,\overline{D} + \overline{E}\,\overline{G} + \overline{E}\,\overline{H}$
  $+ \overline{F}\,\overline{G} + \overline{F}\,\overline{H}$

19. See Figure P–17.

21. See Figure P–18.

23. See Figure P–19.

25. See Figure P–20.

FIGURE P–17

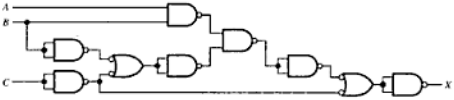
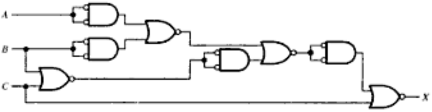
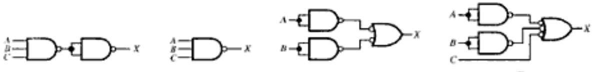
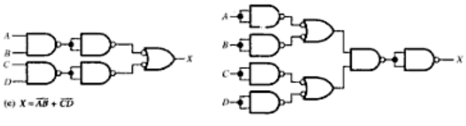


FIGURE P–18



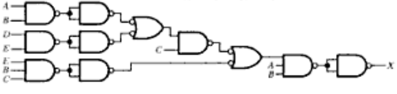


(a) $X = ABC$ (b) $X = \overline{ABC}$ (c) $X = A + B$ (d) $X = A + B + \overline{C}$



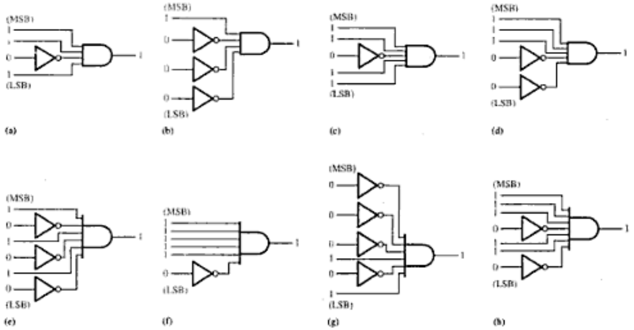
(e) $X = \overline{AB} + \overline{CD}$

(f) $X = (A + B)(C + D)$

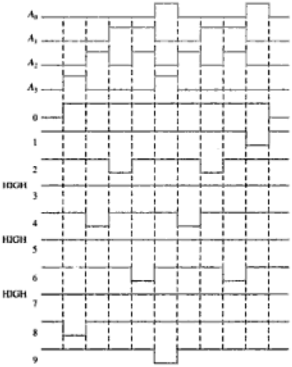


(g) $X = AB[C(\overline{DE} + \overline{AB}) + \overline{BCE}]$

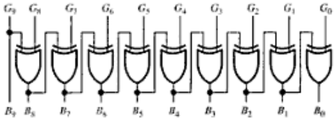
FIGURE P–19


(MSB)
(LSB)
1
0
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

▲ FIGURE P–24

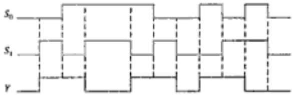

A_0
A_1
A_2
A_3
0
1
2
HIGH
3
4
HIGH
5
6
HIGH
7
8
9

▲ FIGURE P–25


G_9
G_8
G_7
G_6
G_5
G_4
G_3
G_2
G_1
G_0
B_9
B_8
B_7
B_6
B_5
B_4
B_3
B_2
B_1
B_0

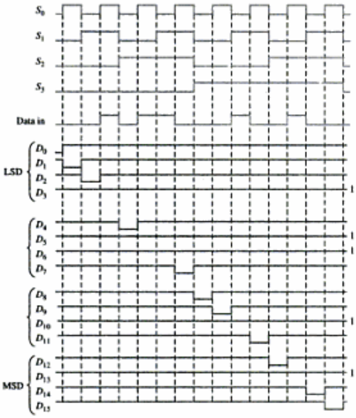
▲ FIGURE P–26

27. See Figure P–27.



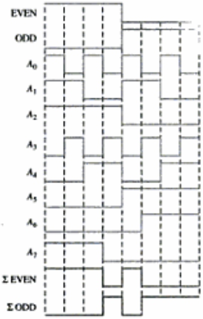
▲ FIGURE P–27

29. See Figure P–28.


S_0
S_1
S_2
S_3
Data in
LSD
D_0
D_1
D_2
D_3
1
D_4
D_5
1
D_6
1
D_7
D_8
D_9
D_{10}
1
D_{11}
MSD
D_{12}
D_{13}
1
D_{14}
D_{15}

▲ **FIGURE P–28**

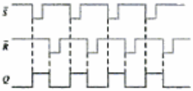
31. See Figure P–29.


EVEN
ODD
A_0
A_1
A_2
A_3
A_4
A_5
A_6
A_7
Σ EVEN
Σ ODD

▲ **FIGURE P–29**

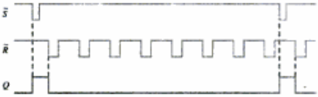
Chapter 7

1. See Figure P–30.



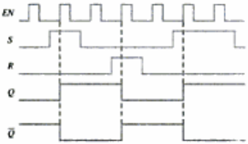
▲ **FIGURE P–30**

3. See Figure P–31.



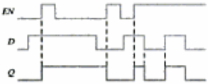
▲ **FIGURE P–31**

5. See Figure P–32.

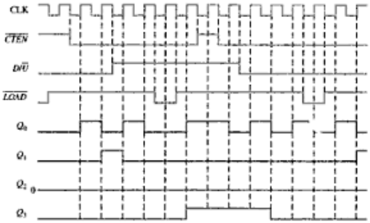


▲ **FIGURE P–32**

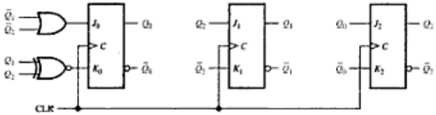
7. See Figure P–33.



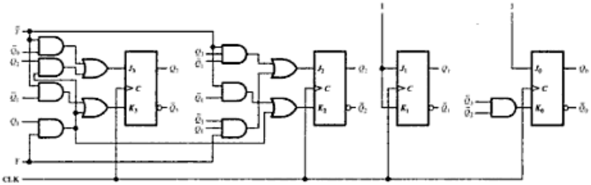
▲ **FIGURE P–33**



▶ **FIGURE P–46**

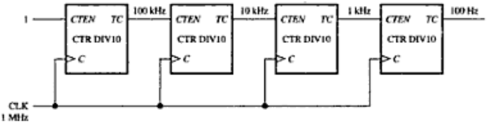

$\bar{Q}_1$
$\bar{Q}_2$
Q_1
Q_2
J_0
C
K_0
Q_0
$\bar{Q}_0$
Q_2
$\bar{Q}_2$
J_1
K_1
Q_1
$\bar{Q}_1$
Q_0
$\bar{Q}_0$
J_2
K_2
Q_2
$\bar{Q}_2$
CLK

▶ **FIGURE P–47**


$\bar{Y}$
Y
CLK
J_3
K_3
Q_3
$\bar{Q}_3$
J_2
K_2
Q_2
$\bar{Q}_2$
J_1
K_1
Q_1
$\bar{Q}_1$
J_0
K_0
Q_0
$\bar{Q}_0$
1
1

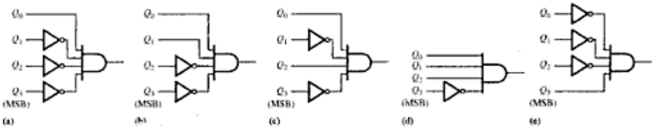
▲ **FIGURE P–48**

21. See Figure P–49 for divide-by-10,000. Add one more DIV10 counter to create a divide-by-100,000.

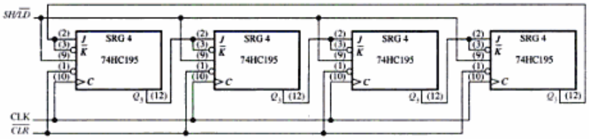

1
CTEN
TC
CTR DIV10
C
100 kHz
10 kHz
1 kHz
100 Hz
CLK
1 MHz

▶ **FIGURE P–49**

23. See Figure P–50.


Q_0
Q_1
Q_2
Q_3
(MSB)
(a)
(b)
(c)
(d)
(e)

▲ **FIGURE P–50**



▲ FIGURE P–60

Chapter 10

1. (a) ROM (b) RAM

3. *Address bus* provides for transfer of address code to memory for accessing any memory location in any order for a read or write operation. *Data bus* provides for transfer of data between the microprocessor and the memory or I/O.

5.

	Bit 0	Bit 1	Bit 2	Bit 3
Row 0	1	0	0	0
Row 1	0	0	0	0
Row 2	0	0	1	0
Row 3	0	0	0	0

7. 512 rows × 128 8-bit columns

9. A SRAM stores bits in flip-flops indefinitely as long as power is applied. A DRAM stores bits in capacitors that must be refreshed periodically to retain the data.

11. See Table P–7.

13. See Figure P–61.

15. Blown links: 1–17, 19–23, 25–31, 34, 37, 38, 40–47, 53, 55, 58, 61, 62, 63, 65, 67, 69

17. Use eight 16k × 4 DRAMs with sixteen address lines. Two of the address lines are decoded to enable the selected memory chips. Four data lines go to each chip.

19. 8 bits, 64k words; 4 bits, 256k words

21. lowest address: $FC0_{16}$
highest address: FFF_{16}

23. A hard disk is formatted into tracks and sectors. Each track is divided into a number of sectors with each sector of a track having a physical address. Hard disks typically have from a few hundred to a few thousand tracks.

25. Magnetic tape has a longer access time than disk because data must be accessed sequentially rather than randomly.

27. Checksum content is in error.

29. (a) ROM 2 (b) ROM 1 (c) All ROMs

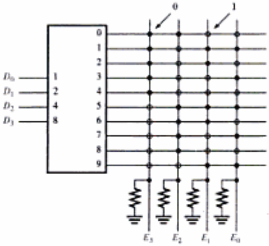
Chapter 11

1. LSI

3. CMOS 5. t_{PLH} = 4.3 ns; t_{PHL} = 10.5 ns

▼ TABLE P–7

INPUTS		OUTPUTS			
A_1	A_0	O_3	O_2	O_1	O_0
0	0	0	1	0	1
0	1	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	0



▲ FIGURE P–61

7. 20 mW

9. No; $V_{OH(min)} < V_{IH(min)}$

11. 0.15 V in HIGH state; 0.25 V in LOW state.

13. Gate *C* 15. 16 ns

17. Gate *C* 19. Yes, G_2

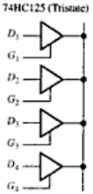
21. (a) on (b) off (c) off (d) on

23. See Figure P–62 for one possible circuit.

25. (a) HIGH (b) Floating
(c) HIGH (d) High-Z

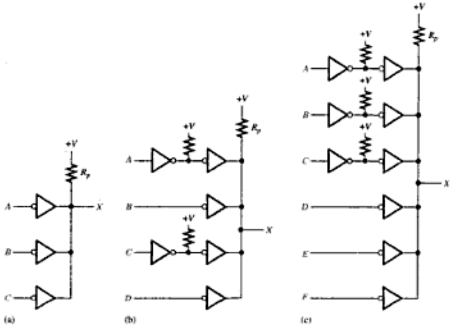
27. (a) LOW (b) LOW (c) LOW

FIGURE P–62



29. See Figure P–63.

FIGURE P–63



31. **(a)** $R_p = 198\ \Omega$
- **(b)** $R_p = 198\ \Omega$
- **(c)** $R_p = 198\ \Omega$

33. ALVC

35. **(a)** A, B to X: 9.9 ns
  C, D to X: 6.6 ns
- **(b)** A to X_1, X_2, X_3: 14 ns
  B to X_1: 7 ns
  C to X_2: 7 ns
  D to X_3: 7 ns
- **(c)** A to X: 11.1 ns
  B to X: 11.1 ns
  C to X: 7.4 ns
  D to X: 7.4 ns

37. ECL operates with nonsaturated BJTs.

Chapter 12

1. ABEL, CUPL

3. $X_1 = \overline{A}B, X_2 = \overline{A}\,\overline{B}, X_3 = A\overline{B}$

5. $X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}C$

7. **(a)** 16 inputs, 2 active-LOW outputs
- **(b)** 12 inputs, 6 active-HIGH outputs
- **(c)** 10 inputs, 8 programmable-polarity outputs
- **(d)** 16 outputs, R is an illegal designation, 6 outputs.

9. $X = A\overline{B}C + \overline{A}\,\overline{C} + B\overline{C} + \overline{B}$

11. Computer, software compiler, programmer

13. Final logic equations, JEDEC file, pinout diagram

15. JEDEC file is a final output file.

17. TDI—Test data in, TDO—Test data out, TCK—test clock, TMS—Test mode select

19. An E^2CMOS cell

GLOSSARY

ABEL Advanced Boolean Expression Language; a software compiler language for SPLD programming; a type of hardware description language (HDL).

access time The time from the application of a valid memory address to the appearance of valid output data.

addend In addition, the number that is added to another number called the augend.

adder A logic circuit used to add two binary numbers.

address The location of a given storage cell or group of cells in a memory; a unique memory location containing one byte.

address bus Generally, a one-way group of conductors from the microprocessor to memory, containing the address information.

adjacency Characteristic of cells in a Karnaugh map in which there is a single-variable change from one cell to another cell next to it on any of its four sides.

alphanumeric Consisting of numerals, letters, and other characters.

amplitude In a pulse waveform, the height or maximum value of the pulse as measured from its low level.

analog Being continuous or having continuous values, as opposed to having a set of discrete values.

analog-to-digital converter (ADC) A device used to convert an analog signal to a sequence of digital codes.

AND A basic logic operation in which a true (HIGH) output occurs only if all the input conditions are true (HIGH).

AND gate A logic gate that produces a HIGH output only when all of the inputs are HIGH.

ANSI American National Standards Institute.

architecture The internal functional arrangement of the elements that give a device its particular operating characteristics.

array In a PLD, a matrix formed by rows of product-term lines and columns of input lines with a programmable cell at each junction.

ASCII American Standard Code for Information Interchange; the most widely used alphanumeric code.

associative law In addition (ORing) and multiplication (ANDing) of three or more variables, the order in which the variables are grouped makes no difference.

asynchronous Having no fixed time relationship; not occurring at the same time.

asynchronous counter A type of counter in which each stage is clocked from the output of the preceding stage.

augend In addition, the number to which the addend is added.

base One of the three regions in a bipolar junction transistor.

BCD Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

BEDO DRAM Burst extended data output dynamic random-access memory.

bidirectional Having two directions. In a bidirectional shift register, the stored data can be shifted right or left.

binary Having two values or states; describes a number system that has a base of two and utilizes 1 and 0 as its digits.

binary sequence Binary numbers occurring in order.

BIOS Basic input/output system; a set of programs in ROM that interfaces the I/O devices in a computer system.

bipolar Having two opposite charge carriers within the transistor structure.

bistable Having two stable states. Flip-flops and latches are bistable multivibrators.

bit A binary digit, which can be either a 1 or 0.

bit time The interval of time occupied by a single bit in a sequence of bits; the period of the clock.

BJT Bipolar junction transistor; a semiconductor device used for switching or amplification. A BJT has two junctions, the base-emitter junction and the base-collector junction.

Boolean addition In Boolean algebra, the OR operation.

Boolean algebra The mathematics of logic circuits.

Boolean expression An expression of variables and operators used to express the operation of a logic circuit.

Boolean multiplication In Boolean algebra, the AND operation.

buffer A circuit that prevents loading of an input or output.

bus A set of interconnections that interface one or more devices based on a standardized specification.

byte A group of eight bits.

cache memory A relatively small, high-speed memory that stores the most recently used instructions or data from the larger but slower main memory.

capacity The total number of data units (bits, nibbles, bytes, words) that a memory can store.

carry The digit generated when the sum of two binary digits exceeds 1.

carry generation The process of producing an output carry in a full-adder when both input bits are 1s.

carry propagation The process of rippling an input carry to become the output carry in a full-adder when either or both of the input bits are 1s and the input carry is a 1.

cascade To connect "end-to-end" as when several counters are connected from the terminal count output of one counter to the enable input of the next counter.

cascading Connecting the output of one device to the input of a similar device, allowing one device to drive another in order to expand the operational capability.

CCD Charge-coupled device; a type of semiconductor memory that stores data in the form of charge packets and is serially accessed.

CD-R CD-Recordable; an optical disk storage device on which data can be stored once.

CD-ROM An optical disk storage device on which data are prestored and can only be read.

CD-RW CD-Rewritable; an optical disk storage on which data can be written and overwritten many times.

cell An area on a Karnaugh map that represents a unique combination of variables in product form; a single storage element in a memory; a fused cross point of a row and column in a PLD; a single storage element in a memory.

character A symbol, letter, or numeral.

circuit An arrangement of electrical and/or electronic components interconnected in such a way as to perform a specified function.

clear An asynchronous input used to reset a flip-flop (make the Q output 0); to place a register or counter in the state in which it contains all 0s.

clock The basic timing signal in a digital system; a periodic waveform in which the interval between pulses equals the time for one bit.

CMOS Complementary metal oxide semiconductor; a class of integrated logic circuits that is implemented with a type of field-effect transistor.

code A set of bits arranged in a unique pattern and used to represent such information as numbers, letters, and other symbols.

collector One of the three regions in a bipolar transistor.

combinational logic A combination of logic gates interconnected to produce a specified Boolean function with no storage or memory capability; sometimes called *combinatorial logic*.

combinational mode The mode of an OLMC in which a combination logic function is implemented.

commutative law In addition (ORing) and multiplication (ANDing) of two variables, the order in which the variables are ORed or ANDed makes no difference.

comparator A digital circuit that compares the magnitudes of two quantities and produces an output indicating the relationship of the quantities.

complement The inverse or opposite of a number; in Boolean algebra, the inverse function, expressed with a bar over the variable. The complement of a 1 is a 0, and vice versa.

counter A digital circuit capable of counting electronic events, such as pulses, by progressing through a sequence of binary states.

CPLD Complex programmable logic device.

CUPL Compiler for Universal Programmable Logic; a software compiler language for PLD programming; a type of hardware description language.

current sinking The action of a circuit in which it accepts current into its output from a load.

current sourcing The action of a circuit in which it sends current out of its output and into a load.

DAT Digital audio tape; a type of magnetic tape format.

data Information in numeric, alphabetic, or other form.

data bus A bidirectional set of conductive paths on which data or instruction codes are transferred into the microprocessor or on which the result of an operation or computation is sent out from the microprocessor.

data selector A circuit that selects data from several inputs one at a time in a sequence and places them on the output; also called a multiplexer.

exponent The part of a floating-point number that represents the number of places that the decimal point (or binary point) is to be moved.

fall time The time interval between the 90% point and the 10% point on the negative-going edge of a pulse.

fan-out The number of equivalent gate inputs of the same family series that a logic gate can drive.

FET Field-effect transistor.

FIFO First in–first out memory.

flash memory A nonvolatile read/write random-access semiconductor memory in which data is stored as charge on the floating gate of a certain FET.

flip-flop A basic storage circuit that can store only one bit at a time; a synchronous bistable device.

floating-point number A number representation based on scientific notation in which the number consists of an exponent and a mantissa.

floppy disk A magnetic storage device; a flexible disk with a diameter of 3.5 inches and a storage capacity of 1.44 Mbytes encased in a rigid plastic housing.

forward bias A voltage polarity condition that allows a semiconductor *pn* junction in a transistor or diode to conduct current.

FPGA Field programmable gate array; a type of PLD, consisting of an array of logic blocks with programmable row and column interconnecting channels surrounded by programmable I/O blocks.

FPM DRAM Fast page mode dynamic random-access memory.

frequency (*f*) The number of pulses in one second for a periodic waveform. The unit of frequency is the hertz.

full-adder A digital circuit that adds two bits and an input carry to produce a sum and an output carry.

fuse The programmable element in certain types of PLDs; also called a fusible link.

GAL Generic array logic; an SPLD with a reprogrammable AND array, a fixed OR array, and programmable output logic macrocells.

gate A logic circuit that performs a specified logic operation, such as AND or OR; one of the three terminals of a field-effect transistor.

glitch A voltage or current spike of short duration, usually unintentionally produced and unwanted.

global cell A programmable cell in an SPLD array that affects all of the OLMCs when programmed.

Gray code An unweighted digital code characterized by a single bit change between adjacent code numbers in a sequence.

half-adder A digital circuit that adds two bits and produces a sum and an output carry. It cannot handle input carries.

hard disk A magnetic disk storage device; typically, a stack of two or more rigid disks enclosed in a sealed housing.

hardware The circuitry and physical components of a computer system (as opposed to the directions called software).

HDL Hardware description language. ABEL and CUPL are examples.

hexadecimal Describes a number system with a base of 16.

high-Z The high-impedance state of a tristate circuit in which the output is effectively disconnected from the rest of the circuit.

hold time The time interval required for the control levels to remain on the inputs to a flip-flop after the triggering edge of the clock in order to reliably activate the device.

hysteresis A characteristic of a threshold-triggered circuit, such as the Schmitt trigger, where the device turns on and off at different input levels.

IEEE Institute of Electrical and Electronic Engineers.

I^2L Integrated injection logic; an IC technology.

increment To increase the binary state of a counter by one.

input The signal or line going into a circuit; a signal that controls the operation of a circuit.

input file The information entered in a computer that describes a logic design using a PLD programming language such as ABEL or CUPL; the part of an ABEL program that contains declarations, logic descriptions, and test vectors.

in-system programming A method for programming SPLDs after they are installed on a printed circuit board.

integer A whole number.

integrated circuit (IC) A type of circuit in which all of the components are integrated on a single chip of semiconductive material of very small size.

interfacing The process of making two or more electronic devices or systems operationally compatible with each other so that they function properly together.

inversion The conversion of a HIGH level to a LOW level or vice versa; also called complementation.

inverter A NOT circuit; a circuit that changes a HIGH to a LOW or vice versa.

Jaz cartridge A magnetic storage device; hard disks encased in a rigid plastic cartridge with storage capacities of 1 Gbyte or 2 Gbytes.

JEDEC Joint Electronic Device Engineering Council; the type of output file for programming SPLDs.

JEDEC file A Joint Electronic Device Engineering Council software file generated from the compiler software that is used by a programming device to implement a logic design in an SPLD; also called a fuse map.

J-K flip-flop A type of flip-flop that can operate in the SET, RESET, no-change, and toggle modes.

Johnson counter A type of register in which a specific prestored pattern of 1s and 0s is shifted through the stages, creating a unique sequence of bit patterns.

JTAG Joint test action group; the IEEE Std. 1149.1 standard interface for in-system programming.

junction The boundary between an *n* region and a *p* region in a BJT.

Karnaugh map An arrangement of cells representing the combinations of literals in a Boolean expression and used for a systematic simplification of the expression.

latch A bistable digital circuit used for storing a bit.

latency period The time it takes for the desired sector to spin under the head once the head is positioned over the desired track of a magnetic hard disk.

LCD Liquid crystal display.

leading edge The first transition of a pulse.

least significant bit (LSB) Generally, the right-most bit in a binary whole number or code.

LED Light-emitting diode.

LIFO Last in–first out memory, memory stack.

literal A variable or the complement of a variable.

load To enter data into a shift register.

local cell A programmable cell in an SPLD array that affects individual OLMCs when programmed.

logic In digital electronics, the decision-making capability of gate circuits, in which a HIGH represents a true statement and a LOW represents a false one.

logic array block (LAB) A group of macrocells that can be interconnected with other LABs or to other I/Os using a programmable interconnect array; also called a function block.

logic element The smallest section of logic in an FPGA that typically contains an LUT, associated logic, and a flip-flop.

look-ahead carry A method of binary addition whereby carries from preceding adder stages are anticipated, thus eliminating carry propagation delays.

LSI Large-scale integration; a level of fixed-function IC complexity in which there are 100 to 9999 equivalent gates per chip.

LUT Look-up table; a memory configuration that is programmed to perform logic functions.

macrocell A section of logic in a CPLD that includes an AND array, a product-term select matrix, an OR gate, and a programmable register section.

magneto-optical disk A storage device that uses electro-magnetism and a laser beam to read and write data.

magnitude The size or value of a quantity.

mantissa The magnitude of a floating-point number.

master-slave flip-flop A type of flip-flop in which the input data are entered into the device on the leading edge of the clock pulses and appear at the output on the trailing edge. Master-slave flip-flops have, for the most part, been replaced by edge-triggered types.

memory array An array of memory cells arranged in rows and columns.

minimization The process that results in an SOP or POS Boolean expression that contains the fewest possible terms with the fewest possible literals per term.

minuend The number from which another number is subtracted.

modulus The number of unique states that a counter will sequence through.

MOS Metal-oxide semiconductor; a type of transistor technology.

MOSFET Metal-oxide semiconductor field-effect transistor.

most significant bit (MSB) The left-most bit in a binary whole number or code.

MSI Medium-scale integration; a level of fixed-function IC complexity in which there are 12 to 99 equivalent gates per chip.

multiplexer (MUX) A circuit (digital device) that switches digital data from several input lines onto a single output line in a specified time sequence.

multiplicand The number that is being multiplied by another number.

multiplier The number that multiplies the multiplicand.

multivibrator A class of digital circuits in which the output is connected back to the input (an arrangement called feedback) to produce either two stable states, one stable state, or no stable states, depending on the configuration.

NAND gate A logic circuit in which a LOW output occurs only if all the inputs are HIGH.

negative-AND An equivalent NOR gate operation in which the HIGH is the active input when all inputs are LOW.

negative-OR An equivalent NAND gate operation in which the HIGH is the active input when one or more of the inputs are LOW.

nibble A group of four bits.

NMOS An *n*-channel metal-oxide semiconductor.

node A common connection point in a circuit in which a gate output is connected to one or more gate inputs.

noise immunity The ability of a circuit to reject unwanted signals.

noise margin The difference between the maximum LOW output of a gate and the maximum acceptable LOW input of an equivalent gate; also, the difference between the minimum HIGH output of a gate and the minimum HIGH input of an equivalent gate.

nonvolatile A term that describes a memory that can retain stored data when the power is removed.

NOR gate A logic gate in which the output is LOW when any or all of the inputs are HIGH.

NOT A basic logic operation that performs inversions.

numeric Related to numbers.

octal Describes a number system with a base of eight.

odd parity The condition of having an odd number of 1s in every group of bits.

OLMC Output logic macrocell; the part of a GAL that can be programmed for either combinational or registered outputs; a block of logic in a GAL that contains a fixed OR gate and other logic for handling inputs and/or outputs.

open-collector A type of output in a logic circuit in which the collector of the output transistor is left disconnected from any internal circuitry and is available for external connection; normally used for driving higher-current or higher-voltage loads.

OR A basic logic operation in which a true (HIGH) output occurs when one or more of the input conditions are true (HIGH).

OR gate A logic gate that produces a HIGH output when any of the inputs is HIGH.

output The signal or line coming out of a circuit.

overflow The condition that occurs when the number of bits in a sum exceeds the number of bits in each of the numbers added.

PAL Programmable array logic; an SPLD with a programmable AND array and a fixed OR array with programmable output logic.

parallel In digital systems, data occurring simultaneously on several lines; the transfer or processing of several bits simultaneously.

parity In relation to binary codes, the condition of evenness or oddness of the number of 1s in a code group.

parity bit A bit attached to each group of information bits to make the total number of 1s odd or even for every group of bits.

period (T) The time required for a periodic waveform to repeat itself.

periodic Describes a waveform that repeats itself at a fixed interval.

PLA Programmable logic array; an SPLD with programmable AND and OR arrays.

PLCC Plastic leaded chip carrier; an SMT package whose leads are turned up under its body in a J-type shape.

PLD Programmable logic device.

PMOS A *p*-channel metal-oxide semiconductor.

positive logic The system of representing a binary 1 with a HIGH and a binary 0 with a LOW.

power dissipation The product of the dc supply voltage and the dc supply current in an electronic circuit; the amount of power required by a circuit.

preset An asynchronous input used to set a flip-flop (make the Q output 1).

priority encoder An encoder in which only the highest value input digit is encoded and any other active input is ignored.

product The result of a multiplication.

product-of-sums (POS) A form of Boolean expression that is basically the ANDing of ORed terms.

variable A symbol used to represent a logical quantity that can have a value of 1 or 0, usually designated by an italic letter.

VLSI Very large-scale integration; a level of IC complexity in which there are 10,000 to 99,000 equivalent gates per chip.

volatile A term that describes a memory that loses its stored data when the power is removed.

weight The value of a digit in a number based on its position in the number.

word A complete unit of binary data.

word capacity The number of words that a memory can store.

word length The number of bits in a word.

WORM Write once-read many; a type of optical storage device.

write The process of storing data in a memory.

zero suppression The process of blanking out leading or trailing zeros in a digital display.

ZIF socket Zero insertion force socket; a type of socket used in most programmers that accepts a PLD package.

Zip disk A type of magnetic storage; a flexible disk with a capacity of 100 Mbytes housed in a rigid plastic cartridge about the size of a floppy.

Index

- ABEL (advanced boolean expression language), 570
- Access time, 399, 412, 434
- ADC (analog-to-digital converter), 2
- Addend, 35
- Adder, 203, 208-218
- Adder expansion, 215
- Addition, 203
- Address, 391
- Address access time, 399
- Address bus, 392
- Address decoder, 392-393
- Address multiplexing, 403
- Address register, 393
- Adjacency, 145
- Alphanumeric codes, 56
- Amplitude, 4
- Analog, 1
- AND array, 505, 533
- AND gate, 10, 82-88, 508
- AND-OR, 173
- AND-OR-Invert, 174
- ANSI/IEEE, 79
- Antifuse, 505, 540
- Architecture, 506
- ASCII, 57-60
- Associative laws, 120-121
- Asynchronous, 281, 305
- Asynchronous counter, 305-312
- Asynchronous SRAM, 395-399
- Augend, 35
- Auto parking control system, 342

- Base, 475
- Baseline, 4
- BCD (binary coded decimal), 51-53, 231, 237
  - addition, 52
  - counter, 309-310, 316, 319
  - decoder, 227
- BEDO DRAM 407
- Biased exponent, 33
- Bidirectional counter, 320-324
- Bidirectional shift register, 368-371
- BiMOS, 487
- Binary, 3, 236
  - adder, 203, 208-218
  - addition, 24, 35, 212
  - counter, 305-311, 313-317
  - data, 7
  - decoder, 222
  - digit, 3, 18
  - division, 40
  - fraction, 19, 33
  - information, 6
  - multiplication, 25
  - number, 18-21
  - point, 19
  - subtraction, 24, 36
- Binary digit, 3, 18
- Bipolar, 453, 475
- Bit, 3, 18, 391
- Bit time, 6
- BJT (bipolar junction transistor), 453, 475
- Bode, George, 9
- Boolean algebra 81, 119-161
  - addition, 91, 119
  - associative laws, 120-121
  - commutative laws, 120-121
  - deMorgan's theorems, 125-129, 185
  - distributive-law, 120-121
  - domain, 135
  - expressions, 81, 86, 91, 98, 103, 141-143, 161, 178
  - laws, 120-121
  - multiplication, 86, 121
  - rules, 122-125
  - simplification 131-134
- Boolean analysis, 129-130
- Borrow, 24
- Buffer, 484, 509, 523
- Burst, 400
- Bus, 397
- Byte, 32, 391

- Cache memory, 400-401
- Carry, 208-209
  - generation, 213
  - propagation, 214
- Cascade, 215
- Cascaded counter, 333-336
- CCD (charge coupled device), 431
- CD (compact disk), 2
- CD player, 2
- CD-R, 438
- CD-ROM, 436
- CD-RW, 438
- Cell, 144, 391, 408
- Cell adjacency, 145, 160
- Cell map, 517
- Checkboard pattern, 441
- Checksum, 440
- Chip, 11
- Clear, 281
- Clock, 6, 273, 289
- CMOS, 453-456, 470-475, 487
- Code converter, 204, 236-238
  - BCD-to-binary, 236
  - binary-to-gray, 237
  - Gray-to-binary, 237
- Codes, 3, 204
- Collector, 475
- Combinational logic, 172-194
- Combinational mode, 522, 529-531
- Commutative laws, 120-121
- Comparator, 203, 218
- Complement, 27, 81, 119, 126
- Complex mode, 531
- Computer, 517
- Contact bounce elimination, 269
- Conventional programming, 517
- Conversion
  - BCD-to-binary, 236
  - BCD-to-decimal, 52
  - binary-to-decimal, 20
  - binary-to-gray, 54
  - binary-to-hexadecimal, 43
  - binary-to-octal,50
  - decimal-to-BCD, 51
  - decimal-to-binary, 21-23
  - decimal-to-hexadecimal, 45
  - decimal-to-octal, 49
  - Gray-to-binary, 55
  - hexadecimal-to-binary, 43
  - hexadecimal-to-decimal, 44
  - octal-to-binary, 49
  - octal-to-decimal, 48
- Count-down chain, 334
- Counter, 207, 293, 304-345
  - asynchronous, 305-312
  - binary, 305-308, 313-315
  - cascaded, 333-336
  - decade, 309-311, 316, 319
  - synchronous, 313-333
  - up/down, 320-324
- Counter decoding, 336-339
- CPLD (complex PLD), 503-504, 533-536
- Current sinking, 468, 481
- Current sourcing, 468, 481

- DAC (digital-to-analog converter), 3
- DAT (digital audio tape), 435
- Data, 7
- Data bus, 392
- Data register, 393
- Data selector, 205, 238-246
- Data sheet, 459-460
- Data storage, 290, 355
- Data transmission system, 250

DC supply voltage, 456, 461
Decade counter, 309-310, 316, 319
Decimal numbers, 17-18, 22, 30
Decimal-to-binary conversion, 21-23, 30
Decoder, 204, 222-231
    address, 392-393
    BCD-to-decimal, 227
    BCD-to-7-segment, 229
    binary, 223
    counter, 336-339
    4-line-to-16-line, 224
    4-line-to-10-line, 227
DeMorgan's theorems, 125-129, 185
Demultiplexer (DEMUX), 205, 247-248
Dependency notation, 241
Design, 324
D flip-flop, 278, 355
Difference, 36
Digital, 1
Digital clock, 340
Digital codes, 54-68
Digital waveform, 4
DIMM, 425
DIP (dual in-line package), 11-12
Disk, 432
Distinctive shape symbol, 79, 82, 88, 93, 99
Distributive law, 120-121
Dividend, 40
Division, 40, 204
Divisor, 40
D latch, 271
DLT (digital linear tape), 436
Documentation file, 519
Domain, 135
Don't care condition, 154
Double precision, 33
Drain, 470
DRAM, 394-395, 401-407
Dual gate symbols, 187
Duty cycle, 5
DVD (digital versatile disk) 438
Dynamic input indicator, 273

ECL (emitter-coupled logic), 453, 488-489, 492
Edge-triggered flip-flop, 273-285
EDO DRAM, 395, 406
E^{2}CMOS, 489-490, 507, 514, 535, 539
EEPROM, 408, 416, 418-419
Emitter, 475
Enable 87, 271
Encoder, 204, 231-235
    decimal-to-BCD, 231
    keyboard, 235
    priority, 233
EPROM 407, 414, 416, 418-419
Equality, 218
Erase, 418
Error correction, 61-68
Error detection, 61-68

Even parity, 61
Exclusive-NOR, 105-107, 177
Exclusive-OR, 103-104, 107, 176
Exponent, 33
Extended ASCII, 59
Extended precision, 33

Falling edge, 4
Fall time, 4
Fan out, 457, 467
Fast page mode DRAM, 395, 403, 406
Feedback, 267
FET (field-effect transistor), 414, 453
FIFO (first in-first out), 426
Fixed-function logic, 453
Flash memory, 416-420
Flip-flop, 206, 273-295, 325-326
Floating gate, 416, 490
Floating-point number, 32, 34
Floppy disk, 434
Flow chart, 439-440
Flow-through SRAM, 399
FPGA (field-programmable gate array), 503-504, 537-542
Fractional number, 23
Frequency, 5, 289
Frequency division, 291, 334
Full-adder, 209, 539
Full-modulus cascading, 335
Function table, 214
Fuse map, 517
Fusible link, 505, 509, 511

GAL (generic array logic), 503, 507, 513-516, 520-532
GAL numbering, 516
Gate, 9, 470
Gated latch, 271-272
Glitch, 251-253, 338
Global cell, 532
Gray code, 54

Half-adder, 208
Hamming code, 63-68
Handling precautions, CMOS, 474
Hard disk, 432-434
HDL (hardware description language), 504
Hertz, 5
Hexadecimal addition, 45
Hexadecimal numbers, 42-48
Hexadecimal subtraction, 46
High-Z state, 473
Hold time, 289

IEEE std. 754-85, 33
IEEE std. 1076-1993, 504
Inequality, 220
Inhibit, 87
Input, 10
In-system programming (ISP), 519

Integer, 32
Integrated circuits, 11-12, 453-492
Integrated circuit packages, 11
Internal carry, 213
Intrusion detection, 92
Invalid code, 51
Inversion, 79, 186
Inverter, 10, 79-81

Jaz, 434
JEDEC file, 517-520
J-K flip-flop, 279
Johnson counter, 371
JTAG, 519-520
Junction, 475

Karnaugh map, 146-162, 326
Keyboard encoder, 380

LAB (logic block array), 533, 540
Lamp test, 230
Laser, 436
Latch, 266-273
Latency period, 434
Leading edge, 4
LED (light emitting diode), 484
LIFO (last in-first out), 428
Literal, 119
Loading, 467
Local cell, 532
Logic, 9-10, 202-207
Logic block, 503, 538
Logic diagram, 176
Logic element, 538, 540
Logic function generator, 243
Logic level, 3, 457, 462-463
Look-ahead carry adder, 213
LSB (least significant bit), 19
LSD (least significant digit), 45
LSI (large scale integration), 453
LUT (look-up table), 537

Macrocell, 533-535
Magnetic storage, 432-438
Magnetic tape, 435
Magneto-optical disk, 436
Magnitude, 203
Mantissa, 32
Mask ROM, 408
Master-slave flip-flop, 285-287
Memory, 207, 390-443
    dynamic, 394-395, 401-407
    flash, 416-420
    magnetic, 432-438
    random access, 207, 394-407
    read only, 207, 393, 407-412
    static, 395-401, 419, 539
Memory address, 392
Memory array, 391, 396-397
Memory capacity, 392
Memory cell, 391, 408

Memory expansion, 420-426
Memory testing, 438
Metal link, 413
Microprocessor, 453
Minimization, 146, 149, 155
Minuend, 36
Modulus, 309
MOSFET, 453, 470
MOS memory, 395, 407
MSBC most significant bit), 19
MSI (medium scale integration), 453
Multiplexer (mux), 205, 238-246, 343
Multiplicand, 38
Multiplication, 38, 204
Multiplier, 38, 204
Multivibrator, 267

NAND gate, 93-98, 183-188, 471, 476
Negation indicator, 79
Negative-AND, 100, 189
Negative logic, 3
Negative-OR, 95, 187
Next-state table, 325
Nibble, 213, 391
NMOS, 453
Noise, 1, 463
Noise immunity, 463
Noise margin, 464
Nondestructive read, 393
Nonperiodic, 5
Nonvolatile memory, 393
NOR gate, 98-103, 183-185, 189
NOT operation, 10, 78-79

Octal numbers, 48-50
Odd parity, 61
OLMC (output logic macrocell), 515-516, 520-532
Open collector gate, 477, 482, 484
Open drain gate, 473
Optical storage, 436-438
OR array, 505
OR gate, 10
Output, 10
Overflow, 36

Page mode, 404
PAL (programmable array logic), 503, 506-513
PAL emulation, 529
PAL numbering, 514
Parallel adder, 211-218
Parallel data, 9, 290, 343
Parallel-to-serial conversion, 343
Parity, 61-68, 249
Parity generator/checker, 248-251
Partial decoding, 309
Partial product, 65
Period, 5
Periodic, 5
PIA (programmable interconnect array), 535
Pin numbering, 11
Pipelined SRAM 399
Pipelining, 399
PLA (programmable logic array), 503, 506-507
PLD (programmable logic device), 502-536
PLD programming, 517-520
PMOS, 489
Polarity indictor, 79
Pop operation, 430
Positive logic, 4
Postponed output symbol, 286
Power dissipation, 456, 465
Powers-of-ten, 17
Powers-of-two, 21
Preset, 281
Priority encoder, 233
Product, 38, 204
Product-of-sums (POS), 137, 141, 155-156
Product term, 120, 136
Programmable array, 504
Programmer, 414, 518
Programming, 414, 417, 504, 517-520
PROM, 407, 412-416, 503, 506
Propagation delay time, 288, 307, 313, 455, 458, 466
Public address system, 2
Pull-up resistor, 235, 482
Pulse, 4
Pulsed operation, 84, 89, 94, 99, 105, 191-194
Pulse train, 5
Pulse triggering, 285
Pulse width, 5, 289
Push operation, 429

QIC (quarter-inch cartridge), 435
Quantization, 2
Quiscent power, 456
Quotient, 40, 204

RAM (random access memory), 207, 394-407
RAM stack, 429
Read, 392-393, 397, 417
Read/write cycle, 397, 404
Read/write head, 432
Real number, 32
Rectangular outline symbol, 79, 82, 88, 95, 101
Recycle, 306
Refresh, 395, 405
Register, 206, 355
Registered mode, 522, 528
Register stack, 428
Remainder, 22, 204
Removable drive, 434-435
Repeated division-by-2 method, 22
Repeated multiplication-by-2 method, 23
Reset, 267
Ring counter, 373
Ripple blanking, 230
Ripple-carry adder, 213
Ripple counter, 307
Rise time, 4
Rising edge, 4
ROM (read only memory), 207, 393, 407-412, 418

Schottky, 479
Sector, 433
Seek time, 434
Semiconductor, 207, 391
Serial data, 9, 343
Serial-to-parallel conversion, 377
Set, 267
Set notation, 267
Set-up time, 288
Seven-segment display, 229, 242
Shaft encoder, 55
Shift register, 206, 354-382
Shift register counter, 371-375
Shorted junction, 413
Sign bit, 29, 33
Signed binary numbers, 29-41
Sign-magnitude, 29-30
Silicon link, 413
SIMM, 425
Simple mode, 529
Single precision numbers, 33
Software, 414
SOP (sum-of-products), 135, 141, 147, 203, 514, 527, 534, 538
Source, 470
Speaker, 2
Speed-power product, 457, 467
SPLD (simple programmable logic device), 503-507
SRAM (static RAM), 394-401, 419, 539-540
S-R flip-flop, 274
S-R latch, 267
SSI (small scale integration), 453
Stack, 428, 433
Stack pointer, 430
Stage, 356
State diagram, 324
State machine, 324
Static memory, 395-401, 419, 539
Storage, 206, 395
Strobing, 253
Subtractor, 204
Subtraction, 24, 36, 204
Subtrahend, 36
Sum, 35, 208
Sum-of-weights, 21
Sum term, 119, 139

Supply voltage, 456, 461
Switching speed, 455, 458
Synchronous, 281
Synchronous burst SRAM 395, 399-401
Synchronous counter, 313-336
Synchronous DRAM, 395, 407
Syntax, 518

Tape, 435
Terminal count, 333
Tied-together inputs, 486
Time delay, 375
Time division multiplexing, 206
Timing diagram, 80, 305
Toggle, 279
Totem-pole output, 484
Track, 433
Trailing edge, 4
Transition table, 325
Tristate logic, 396, 473, 478, 523
Truncated sequence, 309, 335
Truth table, 79
TTL (transistor-transistor logic,) 453, 458, 461, 463, 475-485

UART (universal asynchronous receiver transmitter), 378
ULSI (ultra large scale integration), 453
Unit load, 457, 469
Universal gate, 181-182
Universal shift register, 370
Unused input, 485
Up/down counter, 320-324
UV EPROM 407, 414

Variable 119
Verilog, 504
VHDL, 504
VLSI (very large scale integration), 453
Volatile memory, 393
Voting system, 218

Waveform, 4-6, 80
Weight, 17, 21
Wired-AND, 482
Word, 391
Word-capacity expansion, 424
Word-length expansion, 420
WORM (Write once-read many) disk, 436
Write, 392-393, 397

Zero suppression, 230
ZIF socket, 518
Zip, 434

8421 code, 51
1's complement, 27-29, 31
2's complement, 29-32

Digital Fundamentals

Floyd & Jain

EIGHTH EDITION

The best-selling text, recognized as the authority on the fundamentals of digital systems for nearly a quarter of the century, provides complete, up-to-date coverage from the basic concepts to the programmable logic devices. The book completely covers a semester course on Digital Fundamentals offered to undergraduate students of engineering. Previous knowledge of digital theory and electronics is not a pre-requisite for reading the text. Emphasis has been given to the modern approach followed in industries for the design of digital systems.

Salient Features

- Flash memory, magnetic and optical storage devices
- Programmable Logic Devices (PLDs), Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs)
- Error detection and correction codes
- 317 review questions and 178 self-test questions, with answers
- 178 solved examples supplemented with related problem
- 410 practice problems. Answers to all the odd-numbered problems

Instructors resources available at
www.pearsoned.co.in/thomaslfloyd

Premier12

PEARSON
Education

This edition is manufactured in India and is authorized for sale only
in India, Bangladesh, Bhutan, Pakistan, Nepal, Sri Lanka and the Maldives.

ISBN 978-81-775-8763-0
9788177587630
www.pearsoned.co.in